Vision-Language Models Provide Promptable Representations for Reinforcement Learning

William Chen, Oier Mees, Aviral Kumar, Sergey Levine
U.C. Berkeley

Abstract

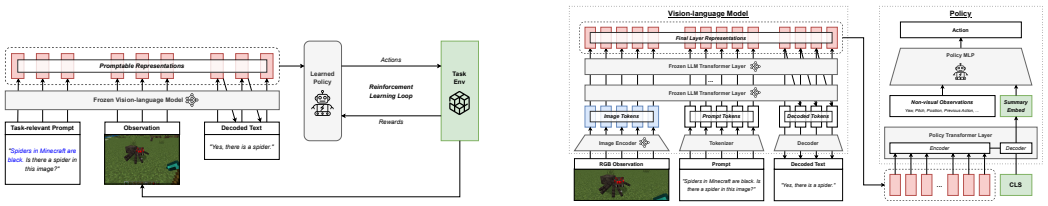
Intelligent beings have the ability to quickly learn new behaviors and tasks by leveraging background world knowledge. We would like to endow RL agents with a similar ability to use contextual prior information. To this end, we propose a novel approach that uses the vast amounts of general-purpose, diverse, and indexable world knowledge encoded in vision-language models (VLMs) pre-trained on Internet-scale data to generate text in response to images and prompts. We initialize RL policies with VLMs by using such models as sources of *promptable representations*: embeddings that are grounded in visual observations and encode semantic features based on the VLM's internal knowledge, as elicited through prompts that provide task context and auxiliary information. We evaluate our approach on RL tasks in Minecraft and find that policies trained on promptable embeddings significantly outperform equivalent policies trained on generic, non-promptable image encoder features and instruction-following methods. In ablations, we find that VLM promptability and text generation both are important in yielding good representations for RL. Finally, we give a simple method for evaluating prompts used by our approach without running expensive RL trials, ensuring that it extracts task-relevant semantic features from the VLM.

1 Introduction

Embodied decision-making often requires representations informed by extensive world knowledge for perceptual grounding, planning, and control. Humans can rapidly learn to perform sensorimotor tasks by drawing on prior knowledge, which might be high-level and abstract ("If I'm cooking something that needs milk, the milk is probably in the refrigerator") or grounded and low-level (e.g., what refrigerators and milk look like). These capabilities would prove highly beneficial for reinforcement learning (RL) too: we aim for our agents to interpret tasks in terms of concepts that can be reasoned about with relevant prior knowledge and grounded with previously-learned representations, thus enabling more efficient learning. However, doing so requires a condensed source of general world knowledge, captured in a form that allows us to specifically index into and access *task-relevant* information. Therefore, we need representations that are contextual, such that agents can use a concise task context to draw out relevant background knowledge, abstractions, and grounded features that aid it in acquiring a new behavior.

One way to do this is with pre-trained foundation models. Transformer-based language models (LMs) and vision-language models (VLMs) are trained on Internet-scale data to enable generalization in downstream tasks requiring facts or common sense. These successes have seen some transfer to embodied control, with (V)LMs being used to reason about goals to produce executable plans [3] or as pre-trained encoders of useful information (like instructions [25] or feedback [37]) that the policy utilizes. Both of these paradigms have major limitations: actions generated by LMs are often not appropriately grounded and (V)LMs are often only suited to producing subtasks, not low-level control signals. On the other hand, using (V)LMs to simply encode inputs under-utilizes their knowledge and reasoning abilities, instead focusing on producing embeddings which reflect language's compositionality. This motivates the development of an algorithm for learning to produce low-level actions that are both grounded and that leverage (V)LMs' knowledge and reasoning.

NeurIPS 2023 Workshop on Robot Learning: Pretraining, Fine-Tuning, and Generalization with Large Scale Models, New Orleans, United States of America



(a) PR2L Instantiation.

(b) PR2L Feature Extraction.

Figure 1: a)**An example instantiation of PR2L.** We query a VLM with a *task-relevant prompt* about observations to produce *promptable representations*, which we train a policy on via RL. Rather than directly asking for actions or specifying the task, the prompt enables *indexing into the VLM's prior world knowledge* to access task-relevant information. They also allow us to inject auxiliary information. b) **Schematic of how we extract task-relevant features from the VLM and use those representations in a policy that we train with RL.** These representations can incorporate task context from the prompt, while generic image encoder representations cannot. As generative Transformers create variable length embeddings, the policy has a Transformer layer that takes in the VLM representations and a "CLS" token to summarize the embeddings.

To this end, we introduce **P**romptable **R**epresentations for **R**einforcement **L**earning (**PR2L**): a flexible framework for guiding vision-language models to produce *semantic features*, which (i) integrate observations with prior task knowledge, and (ii) are grounded into actions via RL (see Figure 1a). We demonstrate our approach in Minecraft [10], as it has semantically-rich and visually-complex tasks found in many practical, realistic, and challenging applications of RL. We find that, by using our approach, we outperform equivalent policies trained on unpromptable visual embeddings or with instruction-conditioning– both popular ways of using pre-trained image models and VLMs respectively for control. Furthermore, we show that promptable representations extracted from general-purpose VLMs can outperform domain-specific representations. Our results and ablations highlight how visually-complex control tasks can benefit from accessing the knowledge captured within VLMs via prompting. We present related works and preliminaries in Appendices A and B.

2 PR2L: Promptable Representations for RL

Our goal is to supplement RL with task-relevant information extracted from VLMs containing general-purpose knowledge. One way to index into this information is by prompting the model to get it to produce semantic information relevant to a given control task. Therefore, our approach queries a VLM with a task-relevant prompt for each visual observation received by the agent, and receives both the decoded text and, critically, the intermediate representations, which we refer to as *promptable representations*. Even though the decoded text might often not be correct or directly actionable, our key insight is that these VLM embeddings can still provide useful semantic features for training control policies via RL. This recipe enables us to incorporate semantic information without the need of re-training or fine-tuning a VLM to directly output actions, as proposed by [5]. Note that our method is *not* an instruction-following method, and it does not require a description of the actual task in natural language. Instead, our approach still learns control via RL, while benefiting from the incorporation of *background context*. Additional details are presented in Appendix C.

Which parts of the network can be used as promptable representations? The VLMs we consider are all based on the Transformer architecture [43], which treats the prompt, input image(s), and decoded text as token sequences. This architecture provides a source of learned representations by computing embeddings for each token at every layer based on the previous layer's token embeddings. In terms of the generative VLM formalism introduced prior, a Transformer-based VLM's representations $_t(I; c; x_{1:t\ \ 1})$ consist of N embeddings per token (the outputs of the N self-attention layers) in the input image I, prompt c, and decoded text $x_{1:t\ \ 1}$. The decoder $p(x_t j\ \ _t)$ extracts the final layer's embedding of the most recent token $x_{t\ \ 1}$, projecting it to a distribution over the token vocabulary and allowing for it to be sampled. When given a visual observation and task prompt, the tokens representing the prompt, image, and answer consequently encode task-relevant semantic information. Thus, for each observation, we use the VLM to sample a response to the task prompt $x_{1:K}$ $p(x_{1:K} j I; c)$. We then use some or all of these token embeddings $_K(I; c; x_{1:t\ \ 1})$ as our promptable representations and feed them, along with any non-visual observation information, as a state representation into our downstream neural-network policy trained with RL. A schematic of our approach is depicted in Figure 1b. Additional design choices are presented in Appendix D.

How do we design good prompts to elicit useful representations from VLMs? As we aim to extract good state representations from the VLM for a downstream policy, we do not use instructions or task descriptions, but task-relevant prompts: questions that make the VLM attend to and encode semantic features in the image that are useful for the RL policy learning to solve the task. For example, if the task is to find a toilet within a house, appropriate prompts include "Is there a toilet in

this image?" and "Am I likely to find a toilet here?" Intuitively, the answers to these questions help determine appropriate actions, making the corresponding representations good for representing the state for a policy. Answering the questions will require the VLM to attend to task-relevant features in the scene, relying on the model's internal conception of what things look like and common-sense semantic relations. Note that prompts based on instructions or task descriptions do not enjoy the above properties: while the goal of those prior methods is to be able to directly query the VLM for the optimal action, the goal of task-relevant prompts is to produce a useful state representation, such that running RL optimization on them can accelerate learning an optimal policy.

Evaluating and optimizing prompts for RL. Since the specific information and representations elicited from the VLM are determined by the prompt, we want to design prompts that produce promptable representations that maximize performance on the downstream task. The brute-force approach would involve running RL with each candidate prompt to measure its efficacy, but this would be computationally very expensive. In lieu of this, we evaluate candidate prompts on a small dataset of observations labeled with semantic features of interest for the considered task. Example features include whether task-relevant entities are in the image, the relative position of said entities, or even actions. We test prompts by querying the VLM and checking how well the resulting decoded text for each image matches ground truth labels. As this is only practical for small, discrete label spaces that are easily expressed in words, we also draw from probing literature [39, 4] and see how well a small model can fit the VLM's embeddings to the labels, thus measuring how extractable said features are from the promptable representations.

3 Experimental Evaluation

We wish to empirically show that one can prompt a VLM to elicit visually-grounded representations that aid in a downstream control task, thus bringing the benefits of Internet-scale VLM pre-training to RL. To this end, we design experiments to answer the following questions: (1) Can promptable representations obtained via task-specific prompts enable more efficient learning than those of pre-trained image encoders? (2) How does PR2L compare to approaches that directly "ask" the VLM to generate the best possible actions for a task specified in the prompt? (3) How well do representations obtained from a general-purpose VLM compare to other domain-specific representations, that are also trained to associate visual observations with text, measured via control performance? For all VLM experiments, we use the InstructBLIP instruction-tuned generative VLM [7].

3.1 Experimental Setup and Comparisons: Minecraft

To answer the questions listed above, we conduct experiments on the Minecraft domain, which provides a number of control tasks that require associating visual observations with rich semantic information to succeed. Moreover, since these observations are distinct from the images in the the pre-training dataset of the VLM, succeeding on these tasks relies crucially on the efficacy of the task-specific prompt in meaningfully affecting the learned representation, enabling us to stress-test our method. For example, while spiders in Minecraft somewhat resemble real-life spiders, they actually exhibit stylistic exaggerations, such as bright red eyes and a large black body. If the task-specific prompt is indeed effective in informing the VLM of these facts, it would produce a representation that is more conducive to policy learning and this would be reflected in task performance.

Minecraft tasks. We consider three Minecraft tasks provided by the MineDojo simulator [10]: (i) combat spider, where the agent must find and defeat a nearby spider while equipped with a shield, diamond sword, and diamond armor; (ii) milk cow, where the agent must milk a nearby cow by using an equipped bucket; and (iii) shear sheep, where the agent must cut wool from a nearby sheep by using equipped shears. We utilize proximal policy optimization (PPO) [36] as our base RL algorithm for all approaches. Additional details are available in Appendix H.

Comparisons. To answer the questions posed at the start of this section, we compare our approach to: (a) methods that do not utilize prompting to obtain representations of the observation, (b) a method that directly "asks" the VLM to output the action to execute on the agent, inspired by the approach of [5], and (c) running RL on the MineCLIP representation [10], which is obtained by fine-tuning CLIP [33] on Minecraft data. Running RL on MineCLIP serves as an "oracle" comparison since this representation was explicitly fine-tuned on a large dataset of Minecraft Youtube videos, whereas our pre-trained VLM is frozen, and is not trained on any Minecraft video data. While [5] also fine-tune the VLM backbone, we are unable to fine-tune this VLM using our computational resources. In order to compensate for this difference, we do not just execute the action from the VLM, but train an RL policy to map this decoded output action into a better action. Finally, comparison (a) does not utilize the task-specific prompt altogether, instead using embeddings from the VLM's image encoder. While this representation of the observation is task-agnostic and still benefits from pre-training, PR2L

utilizes prompting to produce task-specific representations. We utilize the exact same architecture and hyperparameters for this baseline as in PR2L. For more details, see Appendix I.

3.2 Designing Task-Specific Prompts for PR2L

Next, we discuss how to design the task-specific prompts for PR2L. These are not instructions or task descriptions, but prompts that force the VLM to encode semantic information about the task in its representation. The simplest relevant semantic feature for our tasks is the presence of the target entity in a given visual observation. Thus, we choose "Is there a [x] in this image?" as the base of our chosen prompt. We introduce two alternative prompts per task that prepend different amounts of auxiliary information about the target entity. To choose between these prompts, we apply our prompt evaluation strategy by measuring how well the VLM is able to decode the correct answer to the question for annotated images in a small dataset. Full details can be found in Appendix G. We observe that auxiliary text only helps with detecting spiders while systematically degrading the detection of the other two entities. Our results show that this detection success rate is correlated with performance of the RL policy for all tasks. Finally, we define additional prompts for comparison (b), following the recipe for prompt design prescribed by [5]. In these prompts, we also provide a list of allowed actions that the VLM can choose. All prompts are presented in Table 4.

3.3 Results

For all of our results, we report the interquartile mean (IQM) standard error of the returns and successes over 16 seeds per condition for all Minecraft tasks in Figure 2 and the probability of improvement of PR2L over the VLM image encoder baseline in Figure 6, following [2]. For the returns, we apply exponential smoothing to the episode's returns with smoothing factor $= 0.05$. As shown in Figure 2, on all the three tasks, PR2L significantly outperforms both (a) using the VLM image encoder and (b) the method that directly "asks" the VLM for the action, inspired by RT-2. This shows how control tasks can benefit from extracting prior knowledge encoded in VLMs by prompting them with task context and auxiliary information, even in single-task situations where the generalization properties of instruction-following methods do not apply. While PR2L does not outperform the "oracle" MineCLIP policy on combat spider, it performs competitively or better than MineCLIP on the other two tasks that we study, even though the latter is fine-tuned on Minecraft-specific data while InstructBLIP is not. Furthermore, we hypothesize that MineCLIP outperforms PR2L on the spider task because, out of all the entities that we study, Minecraft spiders are the most different visually from real spiders, giving rise to comparatively poor representations in the VLM. Nevertheless, our results in Figure 2 show that PR2L provides an effective approach to transform a general-purpose VLM into a strong task-specific control policy that can often outperform policies trained on domain-specific representations on a given task. We present ablation trials and a discussion in Appendices E and F.

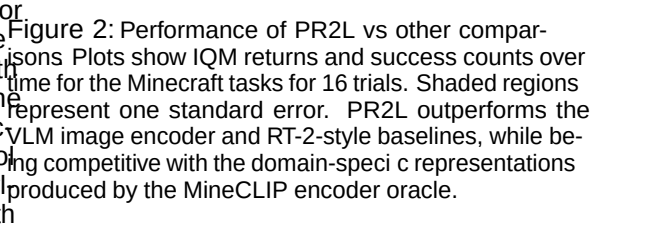


Figure 2: Performance of PR2L vs other comparisons. Plots show IQM returns and success counts over time for the Minecraft tasks for 16 trials. Shaded regions represent one standard error. PR2L outperforms the VLM image encoder and RT-2-style baselines, while being competitive with the domain-specific representations produced by the MineCLIP encoder oracle.

References

[1] Ademi Adeniji, Amber Xie, Carmelo Sferrazza, Younggyo Seo, Stephen James, and Pieter Abbeel. Language reward modulation for pretraining reinforcement learning, 2023.

[2] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G. Bellemare. Deep reinforcement learning at the edge of the statistical precipice, 2022.

[3] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu,

Mengyuan Yan, and Andy Zeng. Do as i can and not as i say: Grounding language in robotic affordances. 2022.

[4] Yonatan Belinkov and James Glass. Analysis methods in neural language processing: A survey, 2019.

[5] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alexander Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Lisa Lee, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, Henryk Michalewski, Igor Mordatch, Karl Pertsch, Kanishka Rao, Krista Reymann, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Pierre Sermanet, Jaspiar Singh, Anikait Singh, Radu Soricut, Huong Tran, Vincent Vanhoucke, Quan Vuong, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Jialin Wu, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. Rt-2: Vision-language-action models transfer web knowledge to robotic control, 2023.

[6] Arthur Bucker, Luis Figueredo, Sami Haddadin, Ashish Kapoor, Shuang Ma, Sai Vemprala, and Rogerio Bonatti. Latte: Language trajectory transformer, 2022.

[7] Wenliang Dai, Junnan Li, Dongxu Li, Anthony Meng Huat Tiong, Junqi Zhao, Weisheng Wang, Boyang Li, Pascale Fung, and Steven Hoi. Instructblip: Towards general-purpose vision-language models with instruction tuning, 2023.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[9] Yuqing Du, Olivia Watkins, Zihan Wang, Cédric Colas, Trevor Darrell, Pieter Abbeel, Abhishek Gupta, and Jacob Andreas. Guiding pretraining in reinforcement learning with large language models, 2023.

[10] Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. In *Neural Information Processing Systems, 2022*, 2022.

[11] Jennifer Hu and Roger Levy. Prompt-based methods may underestimate large language models' linguistic generalizations, 2023.

[12] Chenguang Huang, Oier Mees, Andy Zeng, and Wolfram Burgard. Visual language maps for robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, London, UK, 2023.

[13] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents, 2022.

[14] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning through planning with language models, 2022.

[15] Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. What does BERT learn about the structure of language? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Florence, Italy, 2019. Association for Computational Linguistics.

[16] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12):1–38, mar 2023.

[17] Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hat eld-Dodds, Nova DasSarma, Eli Tran-Johnson, Scott Johnston, Sheer El-Showk, Andy Jones, Nelson Elhage, Tristan Hume, Anna Chen, Yuntao Bai, Sam Bowman, Stanislav Fort, Deep Ganguli, Danny Hernandez, Josh Jacobson, Jackson Kernion,

Shauna Kravec, Liane Lovitt, Kamal Ndousse, Catherine Olsson, Sam Ringer, Dario Amodei, Tom Brown, Jack Clark, Nicholas Joseph, Ben Mann, Sam McCandlish, Chris Olah, and Jared Kaplan. Language models (mostly) know what they know, 2022.

[18] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment anything, 2023.

[19] Belinda Z. Li, Maxwell Nye, and Jacob Andreas. Implicit representations of meaning in neural language models, 2021.

[20] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models, 2023.

[21] Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi. Blip: Bootstrapping language-image pre-training for uni ed vision-language understanding and generation, 2022.

[22] Kenneth Li, Aspen K. Hopkins, David Bau, Fernanda Viégas, Hanspeter P ster, and Martin Wattenberg. Emergent world representations: Exploring a sequence model trained on a synthetic task, 2023.

[23] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control, 2023.

[24] Jessy Lin, Yuqing Du, Olivia Watkins, Danijar Hafner, Pieter Abbeel, Dan Klein, and Anca Dragan. Learning to model the world with language. 2023.

[25] Hao Liu, Lisa Lee, Kimin Lee, and Pieter Abbeel. Instruction-following agents with multimodal transformer, 2023.

[26] Corey Lynch and Pierre Sermanet. Language conditioned imitation learning over unstructured data, 2021.

[27] Arjun Majumdar, Karmesh Yadav, Sergio Arnaud, Yecheng Jason Ma, Claire Chen, Sneha Silwal, Aryan Jain, Vincent-Pierre Berges, Pieter Abbeel, Jitendra Malik, Dhruv Batra, Yixin Lin, Oleksandr Maksymets, Aravind Rajeswaran, and Franziska Meier. Where are we in the search for an arti cial visual cortex for embodied intelligence?, 2023.

[28] Oier Mees, Jessica Borja-Diaz, and Wolfram Burgard. Grounding language with visual affordances over unstructured data. Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), London, UK, 2023.

[29] Vivek Myers, Andre He, Kuan Fang, Homer Walke, Philippe Hansen-Estruch, Ching-An Cheng, Mihai Jalobeanu, Andrey Kolobov, Anca Dragan, and Sergey Levine. Goal representations for instruction following: A semi-supervised language interface to control, 2023.

[30] Suraj Nair, Aravind Rajeswaran, Vikash Kumar, Chelsea Finn, and Abhinav Gupta. R3m: A universal visual representation for robot manipulation, 2022.

[31] Karthik Narasimhan, Regina Barzilay, and Tommi Jaakkola. Grounding language for transfer in deep reinforcement learning, 2018.

[32] Norman Di Palo, Arunkumar Byravan, Leonard Hasenclever, Markus Wulfmeier, Nicolas Heess, and Martin Riedmiller. Towards a uni ed agent with foundation models, 2023.

[33] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021.

[34] Antonin Raf n, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. Journal of Machine Learning Research, 22(268):1–8, 2021.

[35] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2022.

[36] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[37] Pratyusha Sharma, Balakumar Sundaralingam, Valts Blukis, Chris Paxton, Tucker Hermans, Antonio Torralba, Jacob Andreas, and Dieter Fox. Correcting robot plans with natural language feedback. In *Robotics: Science and Systems*, 2022, 2023.

[38] Pratyusha Sharma, Antonio Torralba, and Jacob Andreas. Skill induction and planning with latent language, 2022.

[39] Xing Shi, Inkit Padhi, and Kevin Knight. Does string-based neural MT learn source syntax? In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1526–1534, November 2016.

[40] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Cliport: What and where pathways for robotic manipulation. In *Proceedings of the 5th Conference on Robot Learning (CoRL)*, 2021.

[41] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models, 2022.

[42] Ian Tenney, Dipanjan Das, and Ellie Pavlick. Bert rediscovers the classical nlp pipeline, 2019.

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

[44] Sai Vemprala, Rogerio Bonatti, Arthur Bucker, and Ashish Kapoor. Chatgpt for robotics: Design principles and model abilities. Technical report, Microsoft, 2023.

[45] Gregor Wiedemann, Steffen Remus, Avi Chawla, and Chris Biemann. Does bert make any sense? interpretable word sense disambiguation with contextualized embeddings, 2019.

[46] Andy Zeng, Maria Attarian, Brian Ichter, Krzysztof Choromanski, Adrian Wong, Stefan Welker, Federico Tombari, Aveek Purohit, Michael Ryoo, Vikas Sindhwani, Johnny Lee, Vincent Vanhoucke, and Pete Florence. Socratic models: Composing zero-shot multimodal reasoning with language, 2022.

[47] Andy Zeng, Pete Florence, Jonathan Tompson, Stefan Welker, Jonathan Chien, Maria Attarian, Travis Armstrong, Ivan Krasin, Dan Duong, Vikas Sindhwani, and Johnny Lee. Transporter networks: Rearranging the visual world for robotic manipulation. *Conference on Robot Learning (CoRL)*, 2020.

A Related Works

Embodied (V)LM reasoning. Many recent works have leveraged (V)LMs as embodied reasoners by treating them as priors over effective plans for a given goal. These works use the model's language modeling and auto-regressive generation capabilities to extract such priors as textual subtask sequences [13, 14, 38] or code [23, 41, 46, 44], by effectively using the LM to decompose long-horizon tasks into executable parts or instructions. These systems often need grounding mechanisms to ensure feasibility of their plans (e.g., affordance estimators [3], scene captioners [46], or trajectory labelers [32]). Furthermore, these works often assume access to low-level policies that can execute these subtasks, such as skills to allow a robot to pick up objects [3, 23], which is often a strong assumption. These methods generally do not address how such policies can be acquired, nor how these low-level skills can themselves bene t from the prior knowledge in (V)LMs. Even works in this area that use RL still use (V)LMs as state-dependent priors over reasonable high-level goals to learn [9]. This is a key difference from our work: instead of considering priors on plans or goals, we rely on VLM's implicit knowledge of the world to extract representations which encode task-relevant information. We train a policy to solve the task by converting these features into low-level actions via standard RL, meaning the VLM does not need to know how to take actions for a task.

Embodied (V)LM pre-training. Other works use (V)LMs to embed useful information like instructions [25, 29, 26, 28], feedback [37, 6], reward speci cations [10], and data for world modeling

[24, 31]. These works use (V)LMs as encoders that capture the compositional semantic structure of input text and images, which often aids in generalization: a instruction-conditioned model may never have learned to grasp apples (but was trained to grasp other objects), but by interacting with them in other ways and receiving associated language descriptions, the model might learn what an apple is and its physical properties, thus potentially being able to grasp it zero-shot. In contrast, our method's primary advantage is that the resulting embeddings are informed by world knowledge, both from prompting and pretraining. Rather than just specifying that the task is to acquire an apple, we ask a VLM to parse observations into directly relevant features, like whether there is an apple in the image or if the observed location is likely to contain apples – all information that is useful for RL, even in single-task settings. Thus, we use VLMs to help RL solve new tasks, rather than just to learn how to perform instruction following.

We note these two categories are not binary. For instance, use VLMs to understand instructions, but also reasoning (e.g., guring out the "correct bowl" for a strawberry is one that contains fruits); [32] use a LM to reason about goal subtasks and a VLM to understand when a trajectory matches a subtask description, automating the demonstration collection/labeling of [3], while [1] use a similar framework to pretrain a language-conditioned RL policy that can then be transferred to learning other tasks; and [40] use CLIP to merge vision and text instructions directly into a form that a Transporter [47] policy can operationalize. Nevertheless, these works primarily focus on instruction following in robot manipulation domains. In contrast, our approach prompts a VLM to supplement RL with representations of world knowledge, rather than relying on commands or task speci cations. In addition, except for [1], these works focus on imitation learning, assuming access to existing demonstrations for policy training and ne-tuning, which we forgo by using online RL.

B Preliminaries

Reinforcement learning task and objective. We adopt the standard deep RL partially-observed Markov decision process (POMDP) framework, where the objective is to nd parameters of policy that de nes a distribution over trajectories with maximum expected returns.

Vision-language models. In this work, we utilize generative VLMs (like [21, 20, 7]): models that generate language in response to an image and a text prompt passed as input. This is in contrast to other designs of combining vision and language that either generate images or segmentation [35, 18] and CLIP [33]. Formally, the VLM enables sampling from $p(x_{1:K} | I; c)$, where $x_{1:K}$ represents the K tokens of the output, I is the input image(s), c is the prompt, and p is the distribution over natural language responses produced by the VLM on those inputs. Typically, the VLM is pre-trained on tasks that require building association between vision and language such as image captioning, visual-question answering, or instruction-following. While these differ from the "pure" language modeling objective, all these tasks nonetheless require learning to attend to certain semantic features of input images depending on the given prompt. For auto-regressive generative VLMs, this distribution is factorized as $\prod_t p(x_t | I; c; x_{1:t-1})$. Typical architectures for generative VLMs parameterize these distributions using weights that de ne a representation $\phi_t(I; c; x_{1:t-1})$, which depends on the image I, the prompt c, and the previously emitted tokens, and a decoder $p(x_t | \phi_t(I; c; x_{1:t-1}))$, which de nes a distribution over the next token.

C Motivation for Promptable Representations

Why do we choose to use VLMs in this way, instead of the many other ways of using them for control? In principle, one can directly query a VLM to produce actions for a task given a visual observation. While this may work when high-level goals or subtasks are suf cient, VLMs are empirically bad at yielding the kinds of low-level actions used commonly in RL [38]. As VLMs are mainly trained to follow instructions and answer questions about visual aspects of images, it is more appropriate to use these models to extract semantic features about observations that are conducive to being linked to actions. Speci cally, we elicit features that are useful for the downstream task by querying these VLMs with task-relevant prompts that provide contextual task information, thereby causing the VLM to attend to and interpret appropriate parts of observed images. Extracting these features naïvely by only using the VLMs decoded text has its own challenges: such models often suffer from both hallucinations [15] and an inability to report what they "know" in language, even when their embeddings contain such information [7, 11]. However, even when the text is bad, the

Figure 3: Ablation studies on all Minecraft tasks with the VLM image encoder baseline (blue) and our full approach (red), as shown in Figure 2. All ablations achieve worse performance than PR2L, highlighting the importance of each ablated component (the prompt, VLM generation, or inclusion of auxiliary text). Curves are IQM success counts and shaded regions are the standard error. We apply a third-order Savitsky-Golay filter with window size 10 to improve readability. We present additional metrics in Figure. 5 in the Appendix.

underlying representations still contain valuable granular world information that is potentially lost in the projection to language [19, 45, 12, 22]. Thus, we disregard the generated text in our approach and instead provide our policy the embeddings produced by the VLM in response to prompts asking about relevant semantic features in observations instead.

D Design Choices for Instantiating PR2L

To instantiate our method, several design choices must be made. First, the representations of the VLM's decoded text are dependent on the chosen decoding scheme. E.g., greedy decoding is fast and deterministic, but may yield low-probability decoded tokens; beam search improves on this by considering multiple "branches" of decoded text, at the cost of requiring more compute time (for potentially small improvements); lastly, sampling-based decoding can quickly yield estimates of the maximum likelihood answer, but at the cost of introducing stochasticity, which may increase variance. Given the inherent high-variance of our tasks (due to sparse rewards and partial observability) and the computational expense of VLM decoding, we opt for greedy decoding.

Second, one must choose which VLM layers' embeddings to utilize in the policy. While theoretically, all layers of the VLM could be used, pre-trained Transformer models tend to encode valuable high-level semantic information in their later layers [42, 15]. Thus, we opt to only feed the final two layers' representations into our policy. It's worth noting that unlike conventional fixed-dimensional state representations used in RL, these representation sequences are of variable length. To accommodate this, we incorporate an encoder-decoder Transformer layer in the policy. At each time step in a trajectory, this Transformer receives variable-length VLM representations, which are attended to and converted into a fixed-length summarization by the embeddings of a learned "CLS" token [6] in the decoder (green in Figure 1b). We also note that this policy can receive the observed image directly (e.g., after being tokenized and embedded by the image encoder), so as to not lose any visual information from being processed by the VLM. However, we choose not to do this in our experiments in order to more clearly isolate and demonstrate the usefulness of the VLM's representations in particular.

Finally, while it is possible to fine-tune the VLM for RL end-to-end with the policy, akin to what was proposed by [5], this approach incurs substantial compute, memory, and time overhead, particularly with larger VLMs. Nonetheless, we find that our approach performs better than not using the language and prompting components of the VLM. This holds true even when the VLM is frozen, and only the policy is trained via RL, or when the decoded text occasionally fails to answer the task-specific prompt correctly.

E Ablations

We run several ablation experiments to isolate and understand the importance of various components of PR2L towards extracting good promptable representations for RL. First, we run PR2L no-prompt to see if prompting with task context actually tailors the VLM's generated representations favorably towards the target task, improving over an unprompted VLM. Note that this is not the same

as simply utilizing the image encoder (comparison (a)) alone, since this ablation decodes through the VLM, just with an empty prompt. Second, we run PR2L with our chosen prompts, but no generation of text – i.e., the policy only receives the embeddings associated with the image and prompt (the left and middle red groupings of tokens at the top of Figure 1b, but not the right-most group). This tests the hypothesis that representations of generated text might make certain task-relevant features more salient. For instance, the embeddings for "Spiders in Minecraft are black. Is there a spider in this image?", might not encode the presence of a spider as clearly as if the VLM generates "Yes" in response, impacting downstream performance. Finally, to check if our prompt evaluation and optimization strategy provides a good proxy for downstream task performance while tuning prompts for P2RL, we run PR2L with alternative prompts that were not predicted to be the best, as per our criterion in Appendix G. Concretely, this amounts to removing the auxiliary text from the prompt for *combat spider* and including it for *milk cow* and *shear sheep*.

Results from these ablation experiments are presented in Figure 3. In general, all of these ablations perform worse than PR2L. For *milk cow*, we note the most performant ablation is no generation, perhaps because the generated text is often wrong – among the chosen prompts, it yields the lowest true positive and negative rates for classifying the presence of its corresponding target entity (see Table 1 in Appendix G), though adding auxiliary text makes it even worse, perhaps explaining why *milk cow* experienced the largest performance decrease from adding it back in. Regardless, based on the overall trends, we conclude that (i) the promptable and generative aspects of VLM representations are important for extracting good features for control tasks and (ii) our simple evaluation scheme is an effective proxy measure of how good a prompt is for PR2L.

F Discussion

In this work, we propose Promptable Representations for Reinforcement Learning (PR2L), a method for extracting semantic features from images by prompting VLMs with task context, thereby making use of their extensive general-purpose prior knowledge. We demonstrate this approach in Minecraft, a domain that bene ts from interpreting its visually-complex observations in terms of semantic concepts that can be related to task context. This general framework for using VLMs for control tasks opens many new paths of research. For example, prompts are currently hand-crafted based on the user's conception of useful features for the task. While coming up with effective prompts for our tasks in particular was not dif cult, the process of generating and ef ciently evaluating/optimizing them could be automated, which we leave for future works. Additionally, running PR2L with of ine RL may provide even more in-depth insights into the bene ts of this approach, since it removes the need for exploration (which we do not expect PR2L to help with). Finally, while we consider VLMs as our source of promptable representations, other types of promptable foundation models pre-trained with more sophisticated methods could also be used: e.g., ones trained on videos, domain-speci c data, or even physical interactions might yield even better representations, perhaps which encode physics or action knowledge, rather than just common-sense visual semantics. Developing and using such models with PR2L offers an exciting way to transfer diverse prior knowledge to a broad range of control applications.

G Prompt Evaluation for RL in Minecraft

We discuss how to evaluate prompts to use with PR2L, by showcasing an example for a Minecraft task. We start by noting that the presence and relative location of the entity of interest for each task (i.e., spiders, sheep, or cows) are good features for the policy to have. To evaluate if a prompt elicits these features from the VLM, we collect a small dataset of videos in which each Minecraft entity of interest is on the left, right, middle, or not on screen for the entirety of the clip. Each video is collected by a human player screen recording visual observations from Minecraft of the entity from different angles for around 30 seconds at 30 frames per second (with the exception of the video where the entity is not present, which is a minute long).

We propose prompts that target each of the two features we labeled. First, we evaluate prompts that ask "Is there a(n) [entity] in this image?" As the answers to these questions are just yes/no, we see how well the VLM can directly generate the correct answer for each frame in the collected videos. The VLM should answer "yes" for frames in the three videos where the target entity is on the left, right, or middle of the screen and "no" for the nal video. Second, we evaluate if our prompts can

Target Entity	Prompt	True Positive Rate	True Negative Rate
Spider	“Is there a spider in this image?"	22.27%	100.00%
	“Spiders in Minecraft are black. Is there a spider in this image?"	73.42%	94.54%
	“Spiders in Minecraft are black and have red eyes and long, thin legs. Is there a spider in this image?"	50.50%	99.85%
Cow	“Is there a cow in this image?"	71.00%	45.41%
	“Cows in Minecraft are black and white. Is there a cow in this image?"	98.22%	2.00%
	“Cows in Minecraft are black and white and have four legs. Is there a cow in this image?"	96.67%	7.35%
Sheep	“Is there a sheep in this image?"	88.00%	59.83%
	“Sheep in Minecraft are white. Is there a sheep in this image?"	100.00%	0.00%
	“Sheep in Minecraft are white and have four legs. Is there a sheep in this image?"	100.00%	0.00%

Table 1: InstructBLIP's performance at decoding text indicating that it detected the presence of a target entity when given different prompts. We use this as a proxy metric for prompt engineering for RL, allowing us to determine which prompt to use for PR2L.

extract the entity's relative position (left, right, or middle) in the videos where it is present. We note that the prompts we tried could not extract this feature in the decoded text (e.g., asking “Is the [entity] on the left, right, or middle of the screen?” will always cause the VLM to decode the same text). Thus, we try to see if this feature can be extracted from the decoded texts' representations. We measure this by tting a three-category linear classi er of the entity's position given token-wise meanof the decoded tokens' nal embeddings. This is an unsophisticated and unexpressive classi er, i.e., we do not have to worry about the model potentially memorizing the data, which means that good classi cation performance corresponds to an easy extractability of said feature.

We evaluate three types of prompts per task entity for the rst feature: one simply asking if the entity is present in the image (e.g., “Is there a spider in this image?”) and two others adding varying amounts of auxiliary information about visual characteristics of the entity (e.g., “Spiders in Minecraft are black. Is there a spider in this image?” and “Spiders in Minecraft are black and have red eyes and long, thin legs. Is there a spider in this image?”). We present evaluations of all such prompts in Table 1. We nd that the VLM bene ts greatly from auxiliary information for the spider case only, likely because spiders in Minecraft are the most dissimilar to the ones present in natural images of real spiders, whereas cows and sheep are still comparatively similar, especially in terms of scale and color. However, adding too much auxiliary information degrades performance, perhaps because the input prompt becomes too long, and therefore is out-of-distribution for the types of prompts that the VLM was pre-trained on. This same argument may explain why auxiliary information degrades performance for the other two target entities as well, causing them to almost always answer that said entities are present, even when they are not. Once more, considering that these targets exhibit a higher degree of visual resemblance to to their real counterparts compared to Minecraft spiders, it is reasonable to infer that the VLM would not bene t from auxiliary information. Furthermore, taking into account that the auxiliary information we gave is more common-sense than the information given for the spider, it could imply that the prompts are also more likely to be out-of-distribution (given that “sheep are white” is so obvious that people would not bother expressing it in language), causing the systematic performance degradation.

For the probing evaluation, we nd that all three prompts reach similar nal linear classi abilities for each of their target entities, as shown in Figure 4. While this does not aid in choosing one prompt over another, it does con rm that the VLM's decoded embeddings for each prompt still contains this

valuable and granular position information about the target entity, even though the input prompt did not ask for it

H MineDojo Environment Details

Spaces. The observation space for the Minecraft tasks consists of the following:

1. RGB: Egocentric RGB images from the agent. (160, 256, 3)-size tensor of integers $\in \{0, 1, \ldots, 255\}$.
2. Position: Cartesian coordinates of agent in world frame. 3-element vector of floats.
3. Pitch, Yaw: Orientation of agent in world frame in degrees. Note that we limit the pitch to $15°$ above the horizon to $75°$ below for combat spider, which makes learning easier (as the agent otherwise often spends a significant amount of time looking straight up or down). Two 1-element vectors of floats.
4. Previous Action: The previous action taken by the agent. Set to no operation at the start of each episode. One-hot vector of size $|A| = 53$ for combat spider and 89 otherwise (see below).

This differs from the simplified observation space used in [10] in that we do not use any nearby voxel label information and impose pitch limits for combat spider. This observation space is the same for all Minecraft experiments.

Figure 4: We train a linear classifier to predict the relative position of the target entity (left/right/middle) based on the average VLM embeddings decoded in response to each associated candidate prompt. We find that all three candidate prompts per task elicit embeddings that are similarly highly conducive to this classification scheme.

Actions. The action space is discrete, consisting of 53 or 89 different actions:

1. Turn: Change the yaw and pitch of the agent. The yaw and pitch can be changed up to $90°$ in multiples of $15°$. As they can both be changed at the same time, there are $9 \times 9 = 81$ total different turning actions. The turning action where the yaw and pitch changes are both 0 is the no operation action. Note that, since we impose pitch limits for the spider task, we also limit the change in pitch to $30°$, meaning there are only 45 turning actions in that case.
2. Move: Move forward, backward, left, right, jump up, or jump forward for 6 actions total.
3. Attack: Swing the held item at whatever is targeted at the center of the agent's view.
4. Use Item: Use the held item on whatever is targeted at the center of the agent's view. This is used to milk cows or shear sheep (with an empty bucket or shears respectively). If holding a sword and shield, this action will block attacks with the latter.

Hyperparameter	Task		
	Combat Spider	Milk Cow	Shear Sheep
Total Train Time Steps	150000	100000	100000
Rollout Steps		2048	
Action Entropy Coef cient		5e-3	
Value Function Coef cient		0.5	
Max LR	5e-5	1e-4	1e-4
Min LR	5e-6	1e-4	1e-4
Batch Size		64	
Update Epochs		10	
		0.99	
GAE		0.95	
Clip Range		0.2	
Max Gradient Norm		0.5	
Normalize Advantages		True	

Table 2: PPO hyperparameters for Minecraft tasks, shared by the baselines, our method, and ablations.

This non-combat spider action space is the same as the simpli ed one [10]. All experiments for a given task share the same action space.

World speci cations. MineDojo implements a fast reset functionality that we use. Instead of generating an entirely new world for each episode, fast reset simply respawns the player and all speci ed entities in the same world instance, but with fully restored items, health points, and other relevant task quantities. This lowers the time overhead of resets signi cantly, but also means that some changes to the world (like block destruction) are persistent. However, as breaking blocks generally takes multiple time steps of taking the same action (and does not directly lead to any reward), the agent empirically does not break many blocks aside from tall grass (which is destroyed with a single strike from any held item). We keep all reset parameters (like the agent respawn radius, how far away entities can spawn from the agent, the episode length of 500 etc) at their default values provided by MineDojo.

We stage all tasks in the same area of the same programmatically-generated world: namely, a sun ower plains biome in the world with seed 123. This is the default location for the implementation of the spider combat task in MineDojo. We choose this speci c world/location as it represents a prototypical Minecraft scene with relatively easily-traversable terrain (thus making learning faster and easier).

Additional task details and reward functions. We provide additional notes about our Minecraft tasks.

Combat spider. Upon detecting the agent, the spider approaches and attacks; if the agent's health is depleted, then the episode terminates in failure. The agent receives +1 reward for striking any entity and +10 for defeating the spider. We also include several distractor animals (a cow, pig, chicken, and sheep) that passively wander the task space; the agent can reward game by striking these animals, making credit assignment of success rewards and the overall task harder.

Milk cow. The agent also holds wheat in its off hand, which causes the cow to approach the agent when detected and suf ciently nearby. For each episode, we track the minimum visually-observed distance between the agent and the cow at each time step. The agent receives $+0.1 * |d_{min}|$ reward for decreasing this minimum distance (where $d_{min} < 0$ is the change in this minimum distance at a given time step) and +10 for successfully milking the cow.

Shear sheep. As with milk cow, the agent holds wheat in its off hand to cause the sheep to approach it. The reward function also has the same structure as that task, albeit with different coef cients: $+|d_{min}|$ for decreasing the minimum distance to the sheep and +1.0 for shearing it.

I Policy and Training Details

For our actual RL algorithm, we use the Stable-Baselines3 (version 2.0.0) implementation of clipping-based PPO [34], with hyperparameters presented in Table 2. Many of these parameters are the same

Policy Transformer Hyperparameters	
Transformer Token Size	512
Transformer Feedforward Dim	512
Transformer Number Heads	2
Transformer Number Decoder Layers	1
Transformer Number Encoder Layers	1
Transformer Output Dim	128
Transformer Dropout	0.1
Transformer Nonlinearity	ReLU
Policy MLP Hyperparameters	
Number Hidden Layers	1
Hidden Layer Size	128
Activation Function	tanh
VLM Generation Hyperparameters	
Max Tokens Generated	6
Min Tokens Generated	6
Decoding Scheme	Greedy

Table 3: All policy hyperparameters for all Minecraft tasks.

	PR2L Prompt	RT-2-style Baseline Prompt	Change Auxiliary Text Ablation Prompt
Combat Spider	"Spiders in Minecraft are black. Is there a spider in this image?"	"I want to fight a spider. I can attack, move, or turn. What should I do?"	"Is there a spider in this image?"
Milk Cow	"Is there a cow in this image?"	"I want to milk a cow. I can use my bucket, move, or turn. What should I do?"	"Cows in Minecraft are black and white. Is there a cow in this image?"
Shear Sheep	"Is there a sheep in this image?"	"I want to shear a sheep. I can use my shears, move, or turn. What should I do?"	"Sheep in Minecraft are usually white. Is there a sheep in this image?"

Table 4: Prompts used for querying the VLM with PR2L, comparison (b), and the change auxiliary text ablation. For the last column, we remove the auxiliary text for *combat spider*, and add it in for the other two.

as the ones presented by [10]. For the spider trials, we use a cosine learning rate schedule:

$$\text{LR}(\text{current train step}) = \text{Min LR} + (\text{Max LR} \quad \text{Min LR}) \left(\frac{1 + \cos\left(\frac{\text{current train step}}{\text{total train steps}}\right)}{2}\right) \quad (1)$$

We also present the policy and VLM hyperparameters in Table 3. The hyperparameters and architecture of the MLP part of the policy are primarily defined by the default values and structure defined by the Stable-Baselines3 `ActorCriticPolicy` class. Note that the no generation ablation, VLM image encoder baseline, and MineCLIP trials do not generate text with the VLM, and so all do not use the associated process's hyperparameters. The MineCLIP trials also do not use a Transformer layer in the policy, due to not receiving token sequence embeddings. It instead just uses a MLP, but with two hidden layers (to supplement the lowered policy expressivity due to the lack of a Transformer layer).

Additionally, InstructBLIP's token embeddings are larger than ViT-g/14's (used in the VLM image encoder baseline), and so may carry more information. However, the VLM does not receive any privileged information over the image encoder *from the task environment* – any additional information in the VLM's representations is therefore purely from the model's prompted internal knowledge. Still, to ensure consistent policy expressivity, we include a learned linear layer projecting all representations for this baseline and our approach to the same size (512 dimensions) so that the rest of the policy is the same for both.

J Additional Results and Ablation Plots

We extend Figure 3 with additional performance metrics in Figure 5.

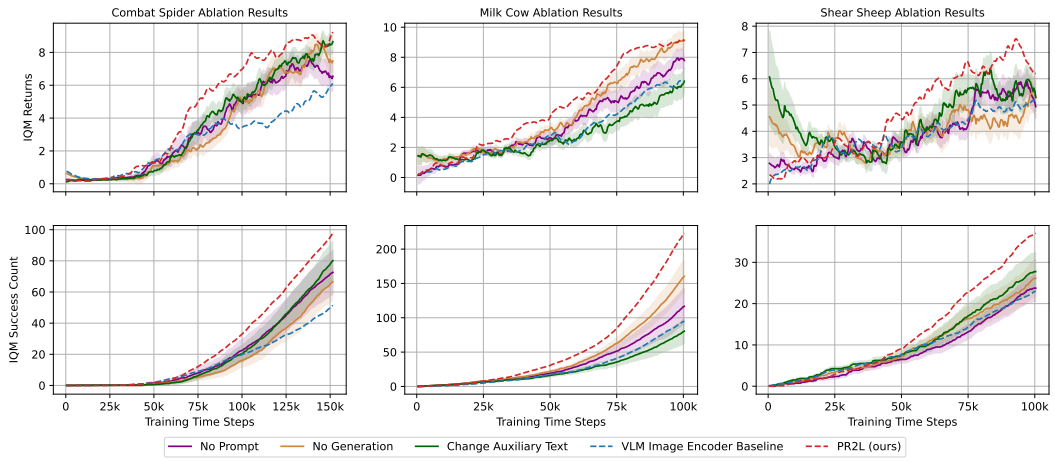


Figure 5: Extended version of Figure 3 with both the returns and success counts for the ablation trials. All curves represent IQMs and shaded regions represent the standard error.

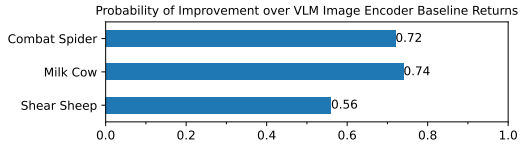


Figure 6: PR2L yields high probability of improvement over the VLM image encoder comparison (a).

K Additional Minecraft Experiments

K.1 Behavior Cloning

We collected expert policy data by training a policy on MineCLIP embeddings to completion on all of our original tasks and saving all transitions to create an offline dataset. We then embedded all transitions with either PR2L or the VLM image encoder. Finally, we train policies with behavior cloning (BC) on successful trajectories under a specified length (300 for *combat spider*, 250 for *milk cow*, and 500 for *shear sheep*) from either set of embeddings for all three tasks, then evaluate their task success rates.

Results are presented in Fig. 7. We first note that, since the expert data was collected from a policy trained on MineCLIP embeddings, the *shear sheep* policy is not very effective (as we found in Fig. 2). Both resulting *shear sheep* BC policies are likewise not very performant. We find that *combat spider* in particular shows a very large gap in performance: the PR2L agent achieves approximately twice the success rate of the VLM image encoder agent *after training for just a single epoch*. The comparatively small amount of training and data necessary to achieve near-expert performance for this task supports our hypothesis that promptable representations from general-purpose VLMs do not help with exploration (they work better in offline cases, where exploration is not a problem), but instead are particularly conducive to being linked to appropriate actions even though the VLM is not producing actions itself. Further investigation of this hypothesis is presented in Appendix L.

K.2 Additional Baselines

We provide additional baselines in *combat spider*, *milk cow*, and *shear sheep*. Specifically, as InstructBLIP's image encoder may not be especially good for control tasks, we train policies on embeddings from VC-1 and R3M – two image encoders with representations that are specifically pretrained for embodied control and decision-making [27, 30]. As they both yield fix-sized embeddings, we use the same policy architecture and task hyperparameters as the MineCLIP baseline experiments. Additionally, to disambiguate whether PR2L is simply more performant due to being able to more reliably detect the task-relevant entity, we train a baseline policy on top of both the VLM image

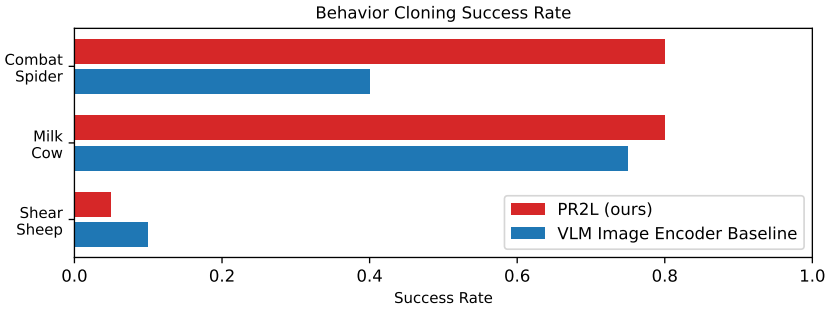


Figure 7: Success rates for BC on either PR2L or VLM image encoder baseline representations for all original tasks. PR2L excels at *combat spider*, even after the policy is trained for a single epoch.

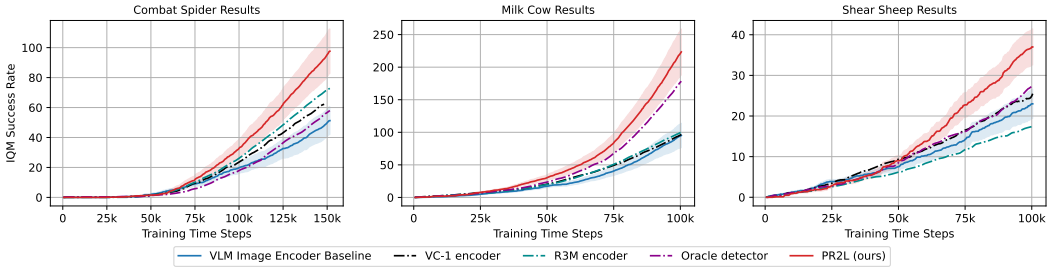


Figure 8: Success counts for the following additional baseline trials: (1) using VC-1 as an encoder, (2) using R3M as an encoder, and (3) using VLM image encoder embeddings with privileged oracle entity detection. All curves represent IQMs and shaded regions represent the standard error (some are omitted for visual clarity).

encoder embeddings and an indicator of whether the entity is in view based on a privileged oracle semantic LIDAR readings (as provided by MineDojo)[1]. This baseline uses the same hyperparameters as the VLM image encoder baseline. We present all results in Fig. 8 and find that PR2L beats all three baselines in all cases.

L Representation Analysis

Why do our prompts yield higher performance than one asking for actions or instruction-following? Intuitively, despite appropriate responses to our task-relevant prompts not directly encoding actions, there should be a strong correlation: e.g., when fighting a spider, if the spider is in view and the VLM detects this, then a good policy should know to attack to get rewards. We therefore wish to investigate if our representations are conducive to easily deciding when certain rewarding actions would be appropriate for a given task – if it is, then such a policy may be more easily learned by RL, which would explain PR2L's improved performance over the baselines.

To investigate this, we use the embeddings of our offline data from the BC experiments (collected by training a MineCLIP encoder policy to high performance on all of our original three tasks, as discussed in Appendix K.1). We specifically look at the embeddings produced by a VLM when given our standard task-relevant prompts and when given the instructions used for our RT-2-style baseline. We then perform principal component analysis (PCA) on the tokenwise average of all embeddings for each observation, thereby projecting the embeddings to a 2D space with maximum variance.

We visualize these low-dimensional space in Fig. 9 for the final 20 successful observations from each task, with the point colors of orange and blue respectively indicating whether the observation results in a functional action (attack or use item) or movement (translation or rotation) by the expert

[1]We note that, as InstructBLIP uses its image encoder's representations as its sole source of visual information, if InstructBLIP *is* actually just doing better object detection, then any information about the presence of the entity must also be available to the VLM image encoder baseline.