Vision-Language Models Provide Promptable Representations for Reinforcement Learning

William Chen [1] **Oier Mees** [1] **Aviral Kumar** [2] **Sergey Levine** [1]

Abstract

Humans can quickly learn new behaviors by leveraging background world knowledge. In contrast, agents trained with reinforcement learning (RL) typically learn behaviors from scratch. We thus propose a novel approach that uses the vast amounts of general and indexable world knowledge encoded in vision-language models (VLMs) pre-trained on Internet-scale data for embodied RL. We initialize policies with VLMs by using them as 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 visually-complex, long horizon RL tasks in Minecraft and robot navigation in Habitat. We find that our policies trained on embeddings extracted from general-purpose VLMs outperform equivalent policies trained on generic, non-promptable image embeddings. We also find our approach outperforms instruction-following methods and performs comparably to domain-specific embeddings.

1. Introduction

Embodied decision-making often requires representations informed by world knowledge for perceptual grounding, planning, and control. Humans 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 be 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 vast amounts of general-purpose 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.

An approach to facilitate this involves integrating RL agents with the prior knowledge and reasoning abilities of 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. Moreover, in-context learning (Brown et al., 2020) and instruction fine-tuning (Ouyang et al., 2022) have provided better ways to index into (V)LMs' knowledge and guide their capabilities based on user needs. These successes have seen some transfer to embodied control, with (V)LMs being used to reason about goals to produce executable plans (Ahn et al., 2022) or as encoders of useful information (like instructions (Liu et al., 2023) or feedback (Sharma et al., 2023)) that the control policy utilizes. Both these paradigms have major limitations: actions generated by LMs are often not appropriately grounded, unless the tasks and scenes are amenable to being expressed or captioned in language. Even then, (V)LMs are often only suited to producing subtask plans, 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 that reflect the compositionality of language (e.g., so an instruction-following policy may generalize). 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.

To this end, we introduce **P**romptable **R**epresentations for **R**einforcement **L**earning (**PR2L**): a flexible framework for guiding VLMs to produce *semantic features*, which (i) integrate observations with prior task knowledge, and (ii) are grounded into actions via RL (see Figure 1). Specifically, we ask a VLM questions about observations that are related to the given control task, encouraging it to attend to task-relevant features in the image based on both its inter-

[1]UC Berkeley [2]Google DeepMind. Correspondence to: William Chen <verityw@berkeley.edu>.

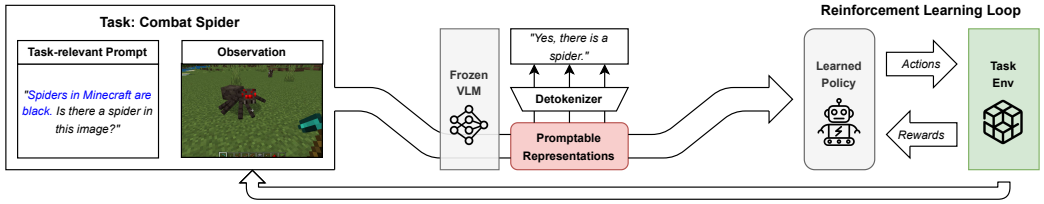

Figure 1. **An example instantiation of PR2L for the *combat spider* Minecraft task.** We query a VLM with a *task-relevant prompt* about observations to produce *promptable representations*, which we train a policy on via RL. Rather than directly asking for actions or specifying the task, the prompt enables *indexing into the VLM's prior world knowledge* to access task-relevant information. This prompt also allows us to inject auxiliary information (e.g., about visual features).

nal world knowledge and any supplemental information injected via prompting. The VLM then encodes this information in decoded text, which is discarded, and associated embeddings, which serve as inputs to a learned policy. In contrast to the standard approach of using pre-trained image encoders to convert visual inputs into *generic* features for downstream learning, our method yields *task-specific* features capturing information particularly conducive to learning a considered task. Thus, the VLM does not just produce an ungrounded encoding of instructions, but embeddings containing semantic information relevant to the given task that is both grounded and informed by the VLM's prior knowledge through prompting.

To the best our knowledge, we introduce the first approach for initializing RL policies with generative VLM representations. We demonstrate our approach on tasks in Minecraft (Fan et al., 2022) and Habitat (Savva et al., 2019), as they have semantically-rich tasks found in many practical, realistic, and challenging applications of RL. We find that PR2L outperforms 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. We also show that promptable representations extracted from general-purpose VLMs are competitive with domain-specific representations. Our results highlight how visually-complex control tasks can benefit from accessing the knowledge captured within VLMs via prompting in both online and offline RL settings.

2. Related Works

Embodied (V)LM reasoning. Many recent works have leveraged (V)LMs 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 (Ahn et al., 2022; Huang et al., 2022b; Sharma et al., 2022) or code (Liang et al., 2023; Singh et al., 2022; Zeng et al., 2022; Vemprala et al., 2023), thereby using the (V)LM to decompose long-horizon tasks into executable parts. These systems often need grounding mechanisms to ensure plan feasibility (e.g., affordance estimators (Ahn et al., 2022), scene captioners (Zeng et al., 2022), or trajectory labelers (Palo et al., 2023)). They also often assume access to low-level policies that can execute these subtasks, such as robot pick-and-place skills (Ahn et al., 2022; Liang et al., 2023), 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 benefit 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 (Du et al., 2023). This is a key difference from our work: instead of considering priors on plans/goals, we rely on VLM's implicit knowledge *of the world* to extract representations which encode task-relevant information. We train a policy to convert 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 (Liu et al., 2023; Myers et al., 2023; Lynch & Sermanet, 2021; Mees et al., 2023; O.M.T. et al., 2023), feedback (Sharma et al., 2023; Bucker et al., 2022), reward specifications (Fan et al., 2022), and data for world modeling (Lin et al., 2023b; Narasimhan et al., 2018). These works use (V)LMs as *encoders* of the compositional semantic structure of input text and images, which aids in generalization: an instruction-conditioned model may never have learned to grasp apples (but can grasp other objects), but by interacting with them in other ways and receiving associated language descriptions, the model might still be able to grasp them zero-shot. In contrast, our method produces embeddings that 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 task-relevant features, like whether there is an apple in the image or if the observed location likely contains apples – information that is useful even in single-task RL. Thus, we use VLMs to help RL solve new tasks, not just to follow instructions.

These two categories are not binary. For instance, Brohan et al. (2023) use VLMs to understand instructions, but also reasoning (e.g., figuring out the "correct bowl" for a strawberry is one that contains fruits); Palo et al. (2023) use a LM to reason about goal subtasks and a VLM to know when a trajectory matches a subtask, automating the demonstra-

tion collection/labeling of Ahn et al. (2022), while Adeniji et al. (2023) use a similar approach to pretrain a language-conditioned RL policy that is transferable to learning other tasks; and Shridhar et al. (2021) use CLIP to merge vision and text instructions directly into a form that a Transporter (Zeng et al., 2020) policy can operationalize. Nevertheless, these works primarily focus on instruction-following for robot manipulation. Our approach instead prompts a VLM to supplement RL with representations of world knowledge, not instructions. In addition, except for Adeniji et al. (2023), these works focus on imitation learning, assuming access to demonstrations for policy learning.

3. Preliminaries

Reinforcement learning. We adopt the standard deep RL partially-observed Markov decision process (POMDP) framework, wherein a given control task is defined by the tuple $(\mathcal{S}, \mathcal{A}, p_T, \gamma, r, \rho_0, \mathcal{O}, p_E)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $p_T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ are the transition probabilities, $\gamma \in (0, 1)$ is the discount factor, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, $\rho_0 : \mathcal{S} \to \mathbb{R}$ is the distribution over initial states, $\mathcal{O}$ is the set of observations (including the visual observations), and $p_E : \mathcal{S} \times \mathcal{O} \to \mathbb{R}$ are observation emission probabilities. The objective is to find parameters θ of policy $\pi_\theta : \mathcal{O} \times \mathcal{A} \to \mathbb{R}$ which, together with ρ_0, p_E, and p_T, defines a distribution over trajectories p_θ with maximum expected returns $\eta(\theta)$:

$$\eta(\theta) = \mathbb{E}_{((s_0,o_0,a_0),(s_1,o_1,a_1),\ldots)\sim p_\theta}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]$$

Vision-language models. In this work, we utilize *generative VLMs* (like (Li et al., 2022; 2023a; Dai et al., 2023)): 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 (Rombach et al., 2022; Kirillov et al., 2023) and CLIP (Radford et al., 2021). 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 define 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 defines a distribution over the next token.

4. 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, PR2L, 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 usable for choosing the action, 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 Brohan et al. (2023). Note that our method is *not* an instruction-following method, and it does not require a description of the actual task. Instead, our approach still learns control via RL, while benefiting from the incorporation of *background context*. In this section, we will describe various components of our approach, accompanied by design choices and other practical considerations.

4.1. 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 sufficient, VLMs are empirically bad at yielding the kinds of low-level actions used commonly in RL (Huang et al., 2022a). 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. We thus 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 VLM's *decoded text* has its own challenges: such models often suffer from both hallucinations (Ji et al., 2023) and an inability to report what they "know" in language, even when their embeddings contain such information (Kadavath et al., 2022; Hu & Levy, 2023). However, even when the text is bad, the underlying *representations* still contain valuable granular world information that is potentially lost in the projection to language (Li et al., 2021; Wiedemann et al., 2019; Huang et al., 2023; Li et al., 2023b). 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.

Which parts of the network can be used as promptable

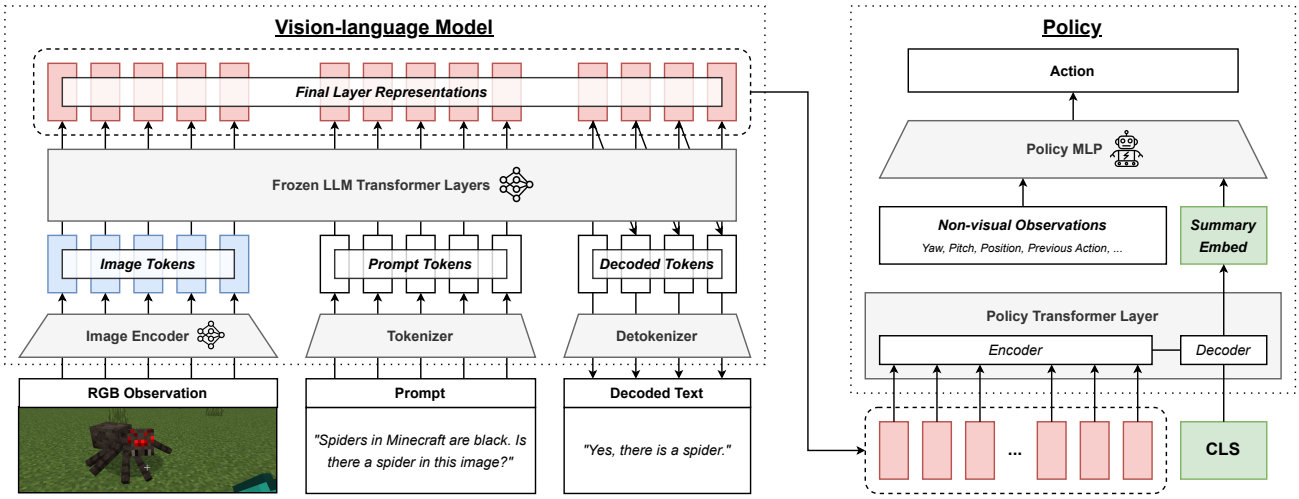

Figure 2. **Schematic of how we extract task-relevant features from the VLM and use them in a policy that we train with RL.** These representations can incorporate task context from the prompt, while generic image embeddings cannot. As generative VLM's embeddings can be variable length, the policy has a Transformer layer that takes in the VLM representations and a "CLS" token, thereby condensing all inputs into a single summary embedding.

representations? The VLMs we consider are all based on the Transformer architecture (Vaswani et al., 2017), 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 $\phi_t(I, c, x_{1:t-1})$ 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 $p(x_t|\phi_t)$ 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 $x_{1:K} \sim p(x_{1:K}|I, c)$. We then use some or all of these token embeddings $\phi_K(I, c, x_{1:t-1})$ as our promptable representations and feed them, along with any non-visual observation information, as a state representation into our neural policy trained with RL.

In summary, our approach involves creating a task-relevant prompt that provides context and auxiliary information. This prompt, alongside the current visual observation from the environment, is fed to into the VLM to generate tokens. While these tokens are used for decoding, they are ultimately discarded. Instead, we utilize the *representations* produced by the VLM (associated with the image, prompt, and decoded text) as input for our policy, which is trained via an off-the-shelf online RL algorithm to produce appropriate actions. A schematic of our approach is depicted in Figure 2.

4.2. Design Choices for Instantiating PR2L

To instantiate our method, several design choices must be made. First, the representations of the VLM's decoded text depend on the chosen decoding scheme: 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 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 (Tenney et al., 2019; Jawahar et al., 2019). Thus, we opt to only feed the final few layers' representations into our policy. As these representation sequences are of variable length, we incorporate an encoder-decoder Transformer layer in the policy. At each time step in a trajectory, this layer receives variable-length VLM representations, which are attended to and converted into a fixed-length summarization by the embeddings of a learned "CLS" token (Devlin et al., 2019) in the decoder (green in Figure 2). 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 do not 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 Brohan et al. (2023), this 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.

4.3. Task-Relevant Prompt

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 (Borja-Diaz et al., 2022). For example, if the task is to find a toilet within a house, appropriate prompts include "What room is this?" and "Am I likely to find a toilet here?" Intuitively, the answers to these questions help determine appropriate actions (e.g., look around the room or explore elsewhere), 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. While the former is not possible without task-specific training data for the VLM in the control task, the latter proves beneficial with off-the-shelf VLMs. Finally, these prompts also provide a place where auxiliary helpful information can be provided: for example, one can describe what certain entities of interest look like, aiding the VLM in detecting them even if they were not commonly found in the model's pre-training data.

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 (if expert demonstrations are available). 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 (Shi et al., 2016; Belinkov & Glass, 2019) 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 (without memorization). While this approach does not directly optimize for task performance, it does act as a proxy that ensures a prompt's resulting representations encode certain semantic features which are helpful for the task.

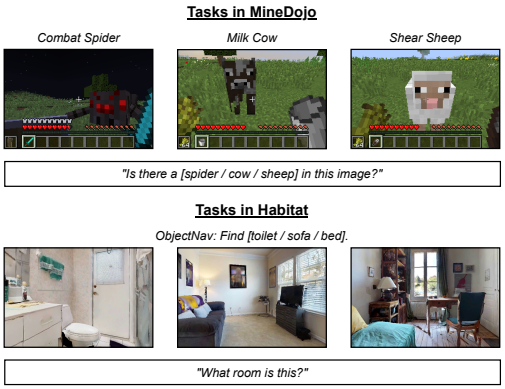

Figure 3. Example tasks, observations, and task-relevant prompts from MineDojo and Habitat.

5. Experimental Evaluation

Our experiments analyze whether promptable representations from VLMs provide benefit to downstream RL and control, thus providing an effective vehicle for transferring Internet-scale knowledge to RL via pre-trained VLMs. We design experiments to answer the following: **(1)** Can promptable representations obtained via task-specific prompts enable more performant and sample-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 does PR2L fare in the offline setting, where exploration (which we do not expect it to help with) is not an issue? In all cases, we use the half-precision, Vicuna-7B version of the InstructBLIP instruction-tuned generative VLM (Dai et al., 2023; Chiang et al., 2023) to produce promptable representations.

5.1. Domain 1: Minecraft

We first conduct experiments in Minecraft, which provides 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. E.g., while spiders in Minecraft somewhat resemble real-life spiders, they 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 all programmatic Minecraft tasks evaluated by Fan et al. (2022): ***combat spider***, ***milk cow***, ***shear sheep***, ***combat zombie***, ***combat enderman***, and ***combat pigman***[1]. The remaining tasks considered by Fan et al. (2022) are creative tasks, which do not have programmatic reward functions or success detectors, so we cannot directly train RL agents on them. We follow the MineDojo definitions of observation/action spaces and reward function structures for these tasks: at each time step, the policy observes an egocentric RGB image, its pose, and its previously action; the policy can choose a discrete action to turn the agent by changing the agent's pitch and/or yaw in discrete increments, move, attack, or use a held item. These tasks are long horizon, with a maximum episode length of 500 - 1000 and taking roughly 200 steps for a learned policy to complete them. See Figure 3 for example observations and Appendix B.1 for more details.

Comparisons. To answer the first two questions from the start of this section, we compare our approach to: **(a)** methods utilizing non-promptable representations of visual observations and **(b)** methods that directly "asks" the VLM to output actions to execute on the agent. Comparison (b) attempts to adapt the approach of Brohan et al. (2023) to our setting and directly outputs the action from the VLM. While Brohan et al. (2023) also fine-tune the VLM backbone, we are unable to do so using our compute resources. To compensate, we do not just execute the action from the VLM, but train an RL policy to map this decoded output action into a better one. Note that, if the VLM already decodes good action texts for the specified task, then simply copying over this action via RL should be easy. Finally, comparison (a) does not utilize the task-specific prompt altogether, instead using task-agnostic embeddings from the VLM's image encoder (specifically, the ViT-g/14 from InstructBLIP – blue in Figure 2). While this representation of the observation still benefits from pre-training, PR2L utilizes prompting to produce *task-specific* representations. For a fair comparison, we use the *exact same* Transformer-layer policy architecture and hyperparameters for this baseline as in PR2L, ensuring that performance differences come from prompting for better representations from the VLM. We use PPO (Schulman et al., 2017) as our base RL algorithm for all Minecraft comparisons. For more details, see Appendix B.2.

[1] Fan et al. (2022) also consider *hunt cow/sheep*. However, we omit them as we were unable to replicate their results on those tasks; all approaches failed to learn them.

5.2. Domain 2: Habitat

PR2L augments state representations with additional useful features, elicited by prompting the VLM. It does not add any sort of exploration incentive. We thus expect our approach to provide the most benefit on tasks that are not bottlenecked by exploration. To highlight this, we run offline RL experiments in the Habitat household simulator. In contrast to Minecraft, tasks in this domain require connecting *natural* images with real-world common sense – namely, about the structure and contents of typical home environments. Moreover, it provides tools for automatically generating high-quality training data for offline learning (Ehsani et al., 2023). Habitat therefore lets us evaluate the feasibility of prompting VLMs for representations of knowledge or useful abstractions about the world that may aid in learning.

Habitat tasks. We consider the ObjectNav suite of tasks in 3D reconstructed household scenes from the HM3D dataset (Savva et al., 2019; Yadav et al., 2023; Ramakrishnan et al., 2021). These tasks involve a simulated robot traversing a home environment to find an instance of a specified object in the shortest path possible. We consider three such objects: **toilets**, **beds**, and **sofas**. We adopt the same reward structure as Yadav et al. (2023), but change the observation space to consist of just RGB vision, previous action, pose, and target object class, omitting depth images to better ensure that experimental performance differences come from the quality of promptable representations vs. unpromptable ones. Like with MineDojo, these tasks are long horizon, taking 80 steps for a privileged shortest path follower to succeed and 200+ for humans. See Figure 3 for example observations and Appendix C for more details.

Comparisons. Our primary comparison is once again between our promptable representations and general-purpose non-promptable ones. We thus repeat the baseline described previously for Minecraft in Section 5.1, training a single agent for all three ObjectNav tasks using both PR2L and the VLM image encoder representations. We empirically note that longer visual embedding sequences tend to perform better in Habitat. To control for this, we opt to use InstructBLIP's Q-Former unprompted embeddings instead of the ViT embeddings directly (which are much longer than PR2L's embedding sequences). As InstructBLIP uses the former representations to extract visual features to be projected into language embedding space, this serves to close the gap in embedding sequence length between our two conditions while still providing us with general visual features that the VLM processes via prompting. We likewise fix all training hyperparameters and offline data to ensure that performance differences come from the quality of state representations, as yielded by prompting. We use CQL with QR-DQN as our offline RL algorithm (Kumar et al., 2020; Dabney et al., 2017). Following Ehsani et al. (2023), we use

Habitat's built-in shortest path follower to generate training data, though add noise to ensure that our dataset contains a mixture of good, mediocre, and failed trajectories. See Appendices C.1 and C.2 for additional details.

5.3. Designing Task-Specific Prompts for PR2L

We now discuss how to design prompts for PR2L. As noted in Section 4.3, 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 feature for our Minecraft tasks is the presence of the target entity in an observation. Thus, we choose "Is there a [target entity] in this image?" as the base of our chosen prompt. We also pick two alternate prompts per task that prepend different amounts of auxiliary information about the target entity. E.g., for *combat spider*, one candidate is "Spiders in Minecraft are black." To choose between these candidate prompts, we measure how well the VLM is able to decode a correct answer to the prompt question of whether or not the target entity is present in the image on a small annotated dataset. Full details of this prompt evaluation scheme for the first three Minecraft tasks are presented in Appendix A and Table 3. We find that auxiliary text only helps with detecting spiders while systematically and significantly degrading the detection of sheep and cows. Our ablations show that this detection success rate metric correlates with performance of the RL policy. Additionally, the prompts used for comparison (b) follow the prompt structure prescribed by Brohan et al. (2023), which motivated this comparison. In these prompts, we also provide a list of actions that the VLM can choose from to the policy. All chosen prompts are presented in Table 1.

For Habitat, we note that ObjectNav would benefit from including the type of room in the state representation, as it informs actions at a high level (e.g., if the agent is looking for a toilet, it is helpful to know when it is in a bathroom). We thus choose "What room is this?" as the prompt for all ObjectNav goal objects.

5.4. Results and Ablations

Minecraft results. We report the interquartile mean (IQM) and standard error number of successes over 16 seeds for all Minecraft tasks in Figure 4. As shown in Figure 4, PR2L tends to outperform both using the VLM image encoder (comparison (a)) and the method that directly "asks" the VLM for the action (comparison (b)) inspired by RT-2. We provide an analysis of why PR2L states are better than RT-2 ones in Appendix E.1. We observe that PR2L embeddings are bimodally distributed, with transitions leading to high reward clustered at one mode. This structure likely enables more efficient learning, thereby showing how control tasks can benefit from extracting prior knowledge encoded in VLMs by prompting them with task context, even in single-task cases where the generalization properties of instruction-following methods do not apply.

Habitat results. We report evaluation success rates and average returns for Habitat ObjectNav in Figure 5. PR2L achieves nearly double the average success rate of the baseline (60.4% vs. 35.2%), supporting the hypothesis that PR2L works especially well when exploration is not needed. Lastly, in Appendix E.2, we find that PR2L causes the VLM to produce highly structured representations that correlate with an expert policy's value function: high-value states are typically labeled by the VLM as being from a room where one would expect to find the target object.

Ablations and additional baselines. We run three ablations on *combat spider*, *milk cow*, and *shear sheep* to isolate and understand the importance of various components of PR2L. First, we run PR2L with *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 just using the image encoder (comparison (a)), as this ablation still decodes through the VLM, just with an empty prompt. Second, we run PR2L with our chosen prompt, 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 in Figure 2, but not the right-most group). This tests the hypothesis that representations of generated text might make certain task-relevant features more salient: e.g., the embeddings for "Is there a cow in this image?", might not encode the presence of a cow as clearly as if the VLM generates "Yes" in response, impacting downstream performance. Finally, to check if our prompt evaluation 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 A. We thus remove the auxiliary text from the prompt for *combat spider* and add it for *milk cow* and *shear sheep*.

We also provide more baselines for these three tasks. First, as InstructBLIP's image encoder may just be bad for control tasks, we run RL on the MineCLIP embeddings (Fan et al., 2022), which is obtained by fine-tuning CLIP (Radford et al., 2021) on Minecraft data. This serves as an "oracle" comparison, as this representation was explicitly fine-tuned on Minecraft YouTube videos, whereas our pre-trained VLM is both frozen and not trained on any Minecraft video data. We also train policies on embeddings from VC-1 and R3M – two image encoders that are specifically pretrained for embodied control (Majumdar et al., 2023; Nair et al., 2022). All three encoders produce a single embedding (not a sequence), so they share an architecture and hyperparameters. Lastly, to disambiguate whether PR2L is simply more performant because it can more reliably detect the task-relevant entity, we train a policy on top of both the VLM image encoder embeddings and an indicator of whether the entity is in view

	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?
Other Combat Tasks	Is there a [target entity] in this image?	I want to fight a [target entity]. I can attack, move, or turn. What should I do?	-

Table 1. Prompts used in Minecraft tasks 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.

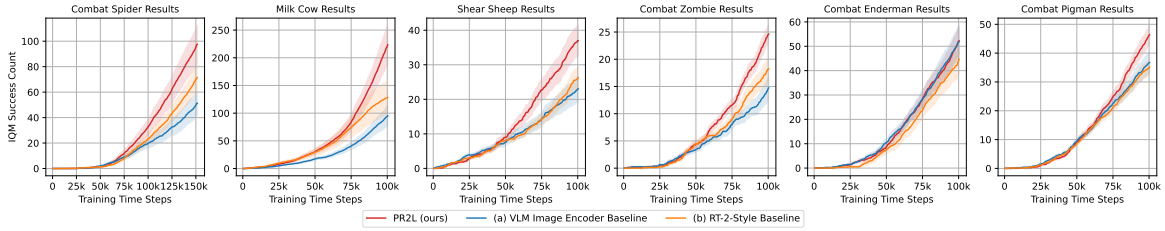

Figure 4. **Performance of PR2L and baselines in Minecraft tasks**. Plots show IQM success counts over time for the Minecraft tasks for 16 trials. Shaded regions are one standard error. PR2L outperforms the VLM image encoder and RT-2-style baselines.

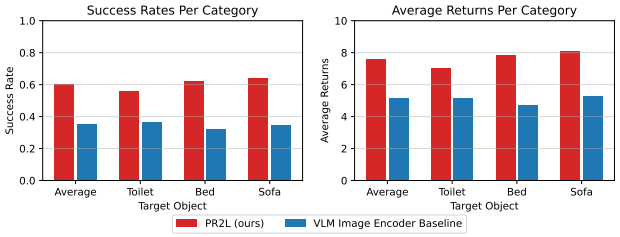

Figure 5. **Offline RL performance of PR2L and baselines in Habitat ObjectNav**. Plots show final evaluation success rates and average returns per target object and overall. PR2L outperforms the baseline in all cases.

based on a privileged oracle semantic detector provided by MineDojo[2]. This baseline uses the same hyperparameters as the VLM image encoder baseline.

Results from these additional experiments are presented in Table 2. In general, all 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 3 in Appendix A), though adding auxiliary text makes it even worse, perhaps explaining why *milk cow* experienced the largest performance decrease from adding it back in. Based on these 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.

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

While PR2L does not outperform the "oracle" MineCLIP policy on *combat spider*, it performs competitively or better than MineCLIP on the other two tasks and beats VC-1, R3M, and the entity detector baselines on all three. 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 (which is trained exclusively on natural images). Still, our results in Table 2 show that PR2L provides an effective approach to transform a general-purpose VLM into a strong task-specific policy (without fine-tuning) that can often outperform policies trained on control-centric representations.

6. Discussion

We propose Promptable Representations for Reinforcement Learning, a method for extracting semantic features from images by prompting VLMs with task context to leverage their extensive general-purpose prior knowledge. We demonstrate PR2L in Minecraft and Habitat, domains that benefit from interpreting observations in terms of semantic concepts that can be related to task context. This framework for using VLMs for control opens many new directions. For example, our prompts are hand-crafted based on the user's conception of useful task features. While coming up with effective prompts for our tasks was not difficult, the process of generating, evaluating, and optimizing them could be automated. Other types of promptable foundation models pre-trained with more sophisticated methods could also be used for PR2L: e.g., ones trained on physical interactions might yield features 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.

Task	PR2L (ours)	VLM Image Encoder Baseline	Ablations			Additional Baselines			
			No Prompt	No Generation	Change Aux. Text	MineCLIP Encoder	VC-1 Encoder	R3M Encoder	Oracle Detector
Combat Spider	97.6 ± 14.9	51.2 ± 9.3	72.6 ± 14.2	66.6 ± 11.8	80.1 ± 12.6	**176 ± 19.8**	62.2 ± 9.4	72.9 ± 8.7	58.0 ± 13.4
Milk Cow	**223.4 ± 35.4**	95.2 ± 18.7	116.6 ± 25.9	160.2 ± 23.6	80.5 ± 17.8	194 ± 33.3	96.6 ± 16.3	100.0 ± 14.1	178.4 ± 42.5
Shear Sheep	**37.0 ± 4.4**	23.0 ± 3.6	23.8 ± 3.2	26.1 ± 4.5	27.8 ± 4.6	23.1 ± 3.7	25.5 ± 3.5	17.5 ± 2.4	27.4 ± 9.3

Table 2. Minecraft ablations and additional baselines with the VLM image encoder baseline and our full approach. All achieve worse performance than PR2L except for MineCLIP encoder for *combat spider*. Values are final IQM success counts and intervals are the standard error. Best performance is bolded per task.

References

Adeniji, A., Xie, A., Sferrazza, C., Seo, Y., James, S., and Abbeel, P. Language reward modulation for pretraining reinforcement learning, 2023.

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

Baker, B., Akkaya, I., Zhokhov, P., Huizinga, J., Tang, J., Ecoffet, A., Houghton, B., Sampedro, R., and Clune, J. Video pretraining (vpt): Learning to act by watching unlabeled online videos, 2022.

Belinkov, Y. and Glass, J. Analysis methods in neural language processing: A survey, 2019.

Borja-Diaz, J., Mees, O., Kalweit, G., Hermann, L., Boedecker, J., and Burgard, W. Affordance learning from play for sample-efficient policy learning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Philadelphia, USA, 2022.

Brohan, A., Brown, N., Carbajal, J., Chebotar, Y., Chen, X., Choromanski, K., Ding, T., Driess, D., Dubey, A., Finn, C., Florence, P., Fu, C., Arenas, M. G., Gopalakrishnan, K., Han, K., Hausman, K., Herzog, A., Hsu, J., Ichter, B., Irpan, A., Joshi, N., Julian, R., Kalashnikov, D., Kuang, Y., Leal, I., Lee, L., Lee, T.-W. E., Levine, S., Lu, Y., Michalewski, H., Mordatch, I., Pertsch, K., Rao, K., Reymann, K., Ryoo, M., Salazar, G., Sanketi, P., Sermanet, P., Singh, J., Singh, A., Soricut, R., Tran, H., Vanhoucke, V., Vuong, Q., Wahid, A., Welker, S., Wohlhart, P., Wu, J., Xia, F., Xiao, T., Xu, P., Xu, S., Yu, T., and Zitkovich, B. Rt-2: Vision-language-action models transfer web knowledge to robotic control, 2023.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

Bucker, A., Figueredo, L., Haddadin, S., Kapoor, A., Ma, S., Vemprala, S., and Bonatti, R. Latte: Language trajectory transformer, 2022.

Cai, S., Wang, Z., Ma, X., Liu, A., and Liang, Y. Open-world multi-task control through goal-aware representation learning and adaptive horizon prediction, 2023.

Chiang, W.-L., Li, Z., Lin, Z., Sheng, Y., Wu, Z., Zhang, H., Zheng, L., Zhuang, S., Zhuang, Y., Gonzalez, J. E., Stoica, I., and Xing, E. P. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023. URL https://lmsys.org/blog/2023-03-30-vicuna/.

Dabney, W., Rowland, M., Bellemare, M. G., and Munos, R. Distributional reinforcement learning with quantile regression, 2017.

Dai, W., Li, J., Li, D., Tiong, A. M. H., Zhao, J., Wang, W., Li, B., Fung, P., and Hoi, S. Instructblip: Towards general-purpose vision-language models with instruction tuning, 2023.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

Ding, Z., Luo, H., Li, K., Yue, J., Huang, T., and Lu, Z. Clip4mc: An rl-friendly vision-language model for minecraft, 2023.

Du, Y., Watkins, O., Wang, Z., Colas, C., Darrell, T., Abbeel, P., Gupta, A., and Andreas, J. Guiding pretraining in reinforcement learning with large language models, 2023.

Ehsani, K., Gupta, T., Hendrix, R., Salvador, J., Weihs, L., Zeng, K.-H., Singh, K. P., Kim, Y., Han, W., Herrasti, A.,

Krishna, R., Schwenk, D., VanderBilt, E., and Kembhavi, A. Imitating shortest paths in simulation enables effective navigation and manipulation in the real world, 2023.

Fan, L., Wang, G., Jiang, Y., Mandlekar, A., Yang, Y., Zhu, H., Tang, A., Huang, D.-A., Zhu, Y., and Anandkumar, A. Minedojo: Building open-ended embodied agents with internet-scale knowledge. In *Neural Information Processing Systems, 2022*, 2022.

Hu, J. and Levy, R. Prompt-based methods may underestimate large language models' linguistic generalizations, 2023.

Huang, C., Mees, O., Zeng, A., and Burgard, W. Visual language maps for robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, London, UK, 2023.

Huang, W., Abbeel, P., Pathak, D., and Mordatch, I. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents, 2022a.

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

Jawahar, G., Sagot, B., and Seddah, D. 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.

Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y. J., Madotto, A., and Fung, P. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12):1–38, mar 2023.

Kadavath, S., Conerly, T., Askell, A., Henighan, T., Drain, D., Perez, E., Schiefer, N., Hatfield-Dodds, Z., DasSarma, N., Tran-Johnson, E., Johnston, S., El-Showk, S., Jones, A., Elhage, N., Hume, T., Chen, A., Bai, Y., Bowman, S., Fort, S., Ganguli, D., Hernandez, D., Jacobson, J., Kernion, J., Kravec, S., Lovitt, L., Ndousse, K., Olsson, C., Ringer, S., Amodei, D., Brown, T., Clark, J., Joseph, N., Mann, B., McCandlish, S., Olah, C., and Kaplan, J. Language models (mostly) know what they know, 2022.

Kanervisto, A., Milani, S., Ramanauskas, K., Topin, N., Lin, Z., Li, J., Shi, J., Ye, D., Fu, Q., Yang, W., Hong, W., Huang, Z., Chen, H., Zeng, G., Lin, Y., Micheli, V., Alonso, E., Fleuret, F., Nikulin, A., Belousov, Y., Svidchenko, O., and Shpilman, A. Minerl diamond 2021 competition: Overview, results, and lessons learned, 2022.

Khanna, M., Mao, Y., Jiang, H., Haresh, S., Shacklett, B., Batra, D., Clegg, A., Undersander, E., Chang, A. X., and Savva, M. Habitat Synthetic Scenes Dataset (HSSD-200): An Analysis of 3D Scene Scale and Realism Tradeoffs for ObjectGoal Navigation. *arXiv preprint*, 2023.

Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., Xiao, T., Whitehead, S., Berg, A. C., Lo, W.-Y., Dollár, P., and Girshick, R. Segment anything, 2023.

Kumar, A., Zhou, A., Tucker, G., and Levine, S. Conservative q-learning for offline reinforcement learning, 2020.

Li, B. Z., Nye, M., and Andreas, J. Implicit representations of meaning in neural language models, 2021.

Li, J., Li, D., Xiong, C., and Hoi, S. Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation, 2022.

Li, J., Li, D., Savarese, S., and Hoi, S. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models, 2023a.

Li, K., Hopkins, A. K., Bau, D., Viégas, F., Pfister, H., and Wattenberg, M. Emergent world representations: Exploring a sequence model trained on a synthetic task, 2023b.

Liang, J., Huang, W., Xia, F., Xu, P., Hausman, K., Ichter, B., Florence, P., and Zeng, A. Code as policies: Language model programs for embodied control, 2023.

Lin, H., Wang, Z., Ma, J., and Liang, Y. Mcu: A task-centric framework for open-ended agent evaluation in minecraft, 2023a.

Lin, J., Du, Y., Watkins, O., Hafner, D., Abbeel, P., Klein, D., and Dragan, A. Learning to model the world with language. 2023b.

Liu, H., Lee, L., Lee, K., and Abbeel, P. Instruction-following agents with multimodal transformer, 2023.

Luo, H., Yue, A., Hong, Z.-W., and Agrawal, P. Stubborn: A strong baseline for indoor object navigation, 2022.

Lynch, C. and Sermanet, P. Language conditioned imitation learning over unstructured data, 2021.

Majumdar, A., Yadav, K., Arnaud, S., Ma, Y. J., Chen, C., Silwal, S., Jain, A., Berges, V.-P., Abbeel, P., Malik, J., Batra, D., Lin, Y., Maksymets, O., Rajeswaran, A., and Meier, F. Where are we in the search for an artificial visual cortex for embodied intelligence?, 2023.

Mees, O., Borja-Diaz, J., and Burgard, W. Grounding language with visual affordances over unstructured data. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, London, UK, 2023.

Myers, V., He, A., Fang, K., Walke, H., Hansen-Estruch, P., Cheng, C.-A., Jalobeanu, M., Kolobov, A., Dragan, A., and Levine, S. Goal representations for instruction following: A semi-supervised language interface to control, 2023.

Nair, S., Rajeswaran, A., Kumar, V., Finn, C., and Gupta, A. R3m: A universal visual representation for robot manipulation, 2022.

Narasimhan, K., Barzilay, R., and Jaakkola, T. Grounding language for transfer in deep reinforcement learning, 2018.

Nottingham, K., Ammanabrolu, P., Suhr, A., Choi, Y., Hajishirzi, H., Singh, S., and Fox, R. Do embodied agents dream of pixelated sheep: Embodied decision making using language guided world modelling, 2023.

O.M.T., Ghosh, D., Walke, H., Pertsch, K., Black, K., Mees, O., Dasari, S., Hejna, J., Xu, C., Luo, J., Kreiman, T., Tan, Y., Sadigh, D., Finn, C., and Levine, S. Octo: An open-source generalist robot policy. https://octo-models.github.io, 2023.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P., Leike, J., and Lowe, R. Training language models to follow instructions with human feedback, 2022.

Palo, N. D., Byravan, A., Hasenclever, L., Wulfmeier, M., Heess, N., and Riedmiller, M. Towards a unified agent with foundation models, 2023.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., and Sutskever, I. Learning transferable visual models from natural language supervision, 2021.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

Ramakrishnan, S. K., Gokaslan, A., Wijmans, E., Maksymets, O., Clegg, A., Turner, J., Undersander, E., Galuba, W., Westbury, A., Chang, A. X., Savva, M., Zhao, Y., and Batra, D. Habitat-matterport 3d dataset (hm3d): 1000 large-scale 3d environments for embodied ai, 2021.

Ramrakhya, R., Batra, D., Wijmans, E., and Das, A. Pirlnav: Pretraining with imitation and rl finetuning for objectnav, 2023.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. High-resolution image synthesis with latent diffusion models, 2022.

Savva, M., Kadian, A., Maksymets, O., Zhao, Y., Wijmans, E., Jain, B., Straub, J., Liu, J., Koltun, V., Malik, J., Parikh, D., and Batra, D. Habitat: A Platform for Embodied AI Research. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms, 2017.

Sharma, P., Torralba, A., and Andreas, J. Skill induction and planning with latent language, 2022.

Sharma, P., Sundaralingam, B., Blukis, V., Paxton, C., Hermans, T., Torralba, A., Andreas, J., and Fox, D. Correcting robot plans with natural language feedback. In *Robotics: Science and Systems, 2022*, 2023.

Shi, X., Padhi, I., and Knight, K. Does string-based neural MT learn source syntax? In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 1526–1534, November 2016.

Shridhar, M., Manuelli, L., and Fox, D. Cliport: What and where pathways for robotic manipulation. In *Proceedings of the 5th Conference on Robot Learning (CoRL)*, 2021.

Singh, I., Blukis, V., Mousavian, A., Goyal, A., Xu, D., Tremblay, J., Fox, D., Thomason, J., and Garg, A. Progprompt: Generating situated robot task plans using large language models, 2022.

Tenney, I., Das, D., and Pavlick, E. Bert rediscovers the classical nlp pipeline, 2019.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need, 2017.

Vemprala, S., Bonatti, R., Bucker, A., and Kapoor, A. Chatgpt for robotics: Design principles and model abilities. Technical report, Microsoft, 2023.

Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., and Anandkumar, A. Voyager: An open-ended embodied agent with large language models, 2023a.

Wang, Z., Cai, S., Chen, G., Liu, A., Ma, X., and Liang, Y. Describe, explain, plan and select: Interactive planning with large language models enables open-world multitask agents, 2023b.

Wiedemann, G., Remus, S., Chawla, A., and Biemann, C. Does bert make any sense? interpretable word sense disambiguation with contextualized embeddings, 2019.

Yadav, K., Krantz, J., Ramrakhya, R., Ramakrishnan, S. K., Yang, J., Wang, A., Turner, J., Gokaslan, A., Berges, V.-P., Mootaghi, R., Maksymets, O., Chang, A. X., Savva, M., Clegg, A., Chaplot, D. S., and Batra, D. Habitat challenge 2023. https://aihabitat.org/challenge/2023/, 2023.

Yuan, H., Zhang, C., Wang, H., Xie, F., Cai, P., Dong, H., and Lu, Z. Plan4mc: Skill reinforcement learning and planning for open-world minecraft tasks, 2023.

Zeng, A., Florence, P., Tompson, J., Welker, S., Chien, J., Attarian, M., Armstrong, T., Krasin, I., Duong, D., Sindhwani, V., and Lee, J. Transporter networks: Rearranging the visual world for robotic manipulation. *Conference on Robot Learning (CoRL)*, 2020.

Zeng, A., Attarian, M., Ichter, B., Choromanski, K., Wong, A., Welker, S., Tombari, F., Purohit, A., Ryoo, M., Sindhwani, V., Lee, J., Vanhoucke, V., and Florence, P. Socratic models: Composing zero-shot multimodal reasoning with language, 2022.

Zhou, B., Li, K., Jiang, J., and Lu, Z. Learning from visual observation via offline pretrained state-to-go transformer, 2023.

Zhu, X., Chen, Y., Tian, H., Tao, C., Su, W., Yang, C., Huang, G., Li, B., Lu, L., Wang, X., Qiao, Y., Zhang, Z., and Dai, J. Ghost in the minecraft: Generally capable agents for open-world environments via large language models with text-based knowledge and memory, 2023.

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 3. 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.

A. 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 final video. Second, we evaluate if our prompts can 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 fitting a three-category linear classifier of the entity's position given the *token-wise mean* of the decoded tokens' final embeddings. This is an unsophisticated and unexpressive classifier, i.e., we do not have to worry about the model potentially memorizing the data, which means that good classification performance corresponds to an easy extractability of said feature.

We evaluate three types of prompts per task entity for the first 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 3. We find that the VLM benefits 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 benefit 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 find that all three prompts reach similar final linear classifiabilities for each of their target entities, as shown in Figure 6. While this does not aid in choosing one prompt over another, it does confirm 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*.

B. MineDojo Details

B.1. 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 $\in \{0, 1, ..., 255\}$.
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 $|\mathcal{A}| = 53$ for *combat spider* and 89 otherwise (see below).

This differs from the simplified observation space used in (Fan et al., 2022) 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.

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 $\pm 90°$ 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 $\pm 30°$, 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.

This non-*combat spider* action space is the same as the simplified one in (Fan et al., 2022). All experiments for a given task share the same action space.

World specifications. 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 specified 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 significantly, 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, etc) at their default values provided by MineDojo.

We stage all tasks in the same area of the same programmatically-generated world: namely, a sunflower 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 specific 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 sufficiently nearby. For each episode, we track the minimum visually-observed distance between the agent and the cow at each time step. The agent receives $+0.1|\Delta d_{\text{min}}|$ reward for decreasing this minimum distance (where $\Delta d_{\text{min}} \leq 0$ 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 coefficients: $+|\Delta d_{\text{min}}|$ for decreasing the minimum distance to the sheep and $+10$ for shearing it.

Combat zombie: Same as *combat spider*, but the enemy is a zombie. We increase the episode length to 1000, as the zombie has more health points than the spider.

Combat enderman: Same as *combat spider*, but the enemy is an Enderman. As with combat zombie, we increase the episode length to 1000. Note that Endermen are non-hostile (until directly looked at for sufficiently long or attacked) and have significantly more health points than other enemies. We thus enchant the agent's sword to deal more damage and decrease the initial spawn distance of the enderman from the agent.

Combat pigman: Same as *combat spider*, but the enemy is a hostile zombie pigman. As with combat zombie, we increase the episode length to 1000.

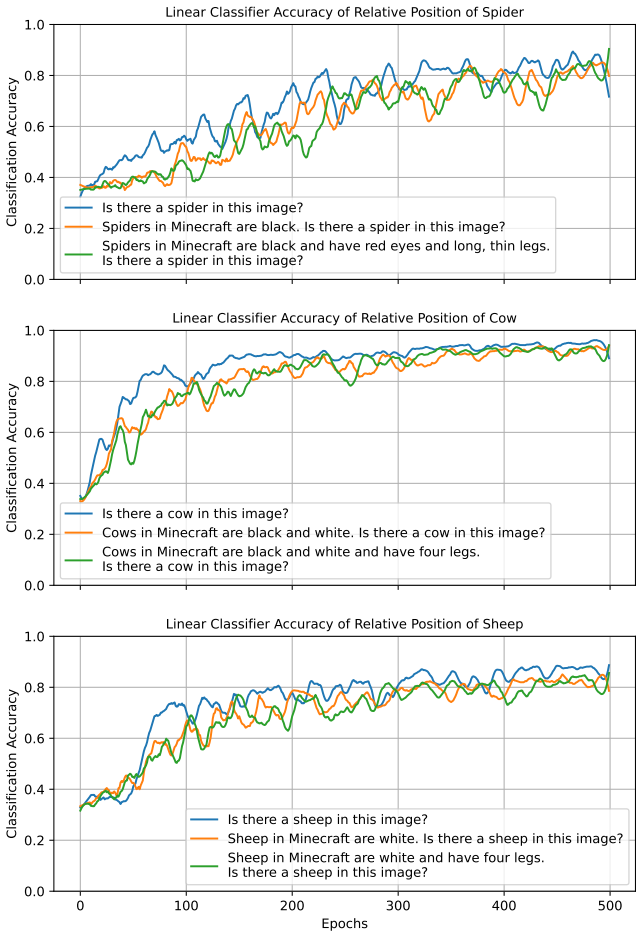

Figure 6. We train a linear classifier to predict the relative position of the target entity (left/right/middle) based on the average VLM embeddings decoded in response to each associated candidate prompt. We find that all three candidate prompts per task elicit embeddings that are similarly highly conducive to this classification scheme.

Hyperparameter	Task					
	Combat Spider	*Milk Cow*	*Shear Sheep*	*Combat Zombie*	*Combat Enderman*	*Combat Pigman*
Total Train Steps	150000			100000		
Rollout Steps				2048		
Action Entropy Coefficient				5e-3		
Value Function Coefficient				0.5		
Max LR	5e-5	1e-4	1e-4	5e-5	1e-4	5e-5
Min LR	5e-6	1e-4	1e-4	5e-6	1e-4	5e-6
Batch Size				64		
Update Epochs				10		
γ				0.99		
GAE λ				0.95		
Clip Range				0.2		
Max Gradient Norm				0.5		
Normalize Advantage				True		

Table 4. PPO hyperparameters for Minecraft tasks, shared by the baselines, our method, and ablations.

Policy Transformer Hyperparameters	
Transformer Token Size	512 / 128
Transformer Feedforward Dim	512 / 128
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 5. All policy hyperparameters for all Minecraft tasks. Smaller token sizes and feedforward dimensions are used for *combat [zombie/enderman/pigman]*.

B.2. Policy and Training Details

For our actual RL algorithm, we use the Stable-Baselines3 (version 2.0.0) implementation of clipping-based PPO (Raffin et al., 2021), with hyperparameters presented in Table 4. Many of these parameters are the same as the ones presented by (Fan et al., 2022). For the spider trials, we use a cosine learning rate schedule:

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

We also present the policy and VLM hyperparameters in Table 5. 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.

C. Habitat Details

C.1. Environment Details

The spaces and agent/task specifications are largely the same as the defaults provided by Habitat, as specified in the HM3D ObjectNav configuration file (Savva et al., 2019).

Spaces. The observation space for Habitat consists of the following:

1. **RGB:** Egocentric RGB images from the agent. (480, 640, 3)-size tensor of integers $\in \{0, 1, ..., 255\}$. By default, agents also receive depth images, but we remove them to ensure that state representations are grounded primarily in visual observations.
2. **Position:** Horizontal Cartesian coordinates of agent. 2-element vector of floats.
3. **Compass:** Yaw of the agent. Single 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 $|\mathcal{A}| = 4$.
5. **Object Goal:** Which object the agent is aiming to find. One-hot vector of size 3.

The action space is the standard Habitat-Lab action space, though we remove the pitch-changing actions, leaving only four:

1. **Turn:** Turn left or right, changing the yaw by $30°$.
2. **Move Forward:** Move forward a fixed amount or until the agent collides with something.
3. **Stop:** Ends the episode, indicating that the agent believes it has found the goal object.

All observations, actions, and associated dynamics are deterministic.

World specifications. In ObjectNav, an agent is spawned in a household environment and must find and navigate to an instance of a specified target object in as efficient a path as possible. Doing so effectively requires a common-sense understanding of where household objects are often found and the structure of standard homes.

We find that most Habitat learning algorithms require either significant system engineering (e.g., semantic SLAM systems, like in Luo et al. (2022)) or very large training datasets (Ramrakhya et al., 2023; Ehsani et al., 2023; Khanna et al., 2023). Such works also often use more complex policies that incorporate histories of observations and actions, either via recurrent or Transformer architectures. To better study the effects of promptability on representations, we choose to simplify the ObjectNav task in several ways.

We train and evaluate on ObjectNav episodes from the Habitat-Matterport 3D v2 dataset's validation split (Ramakrishnan et al., 2021). We choose said dataset because its environments are the most realistic, being generated from real-world 3D scans. Likewise, they offer better visual fidelity than simulated counterparts (despite sometimes having reconstruction artifacts), allowing VLMs trained on naturalistic images to better parse observations. We pick 32 reconstructed 3D home environments with at least one instance of each of the three target objects (toilet, bed, and sofa) and an annotator quality score of at least 4 out of 5. We choose to remove *plants* and *televisions* from the goal object set due to finding numerous unlabeled instances of them. Additionally, we remove chairs, as they are significantly more common than other goal objects and thus usually can be found in much shorter episodes. This simplified problem formulation enables us to remove many of the "tricks" that aid ObjectNav, such as using omnidirectional views or policies with history; our agent makes action decisions purely based on its current visual observation and pose, allowing us to do "vanilla" RL to better isolate the effect of PR2L.

To generate data, we use Habitat's built-in greedy shortest geodesic path follower. Imitating such demonstrations allows policies to learn unintuitively emergent and performant navigation behaviors (Ehsani et al., 2023) at scale. For each defined starting location in our considered households, we autonomously collect data by using the path follower to navigate to each reachable instance of the corresponding goal object. This yields high quality, near-optimal data. We then supplement our dataset by generating lower-quality data. Specifically, for each computed near-optimal path from a starting location to a goal object instance, we choose to inject action noise partway through the trajectory (uniformly at random from $0 - 90\%$ of the way through). At that point, all subsequent actions have a $0 - 50\%$ probability (again chosen uniformly at random) of being a random action other than the one specified by the path follower. To ensure that paths are sufficiently long, we choose to make the probability of choosing the stop action 10% and the other two movement actions 45%. In total, we collect 107518 observations over 2364 trajectories.

Reward functions. The ObjectNav challenge evaluates agents based on the average "success weighted by path length" (SPL) metric (Yadav et al., 2023): if an agent succeeds at taking the *stop* action while close to an instance of the goal object, it gets $SPL(p, l) = \frac{l}{\max(l,p)}$ points, where l is the actual shortest path from the starting point to an instance of the goal object and p is the length of the path that the agent actually took during that particular episode. If the agent stops while not close to the target object, the SPL is 0. Thus, taking the most efficient path to the nearest goal object and stopping yields a maximum SPL of 1.

We use this to design our reward function. Specifically, when the agent stops, it receives a reward of $+10SPL(p, l)$. Additionally, we add a shaping reward of the change in geodesic distance to the nearest goal object instance each time the agent moves (where lowering that distance yields a positive reward).

C.2. Policy and Training Details

For our offline RL experiments in Habitat, we use Conservative Q-Learning (CQL) on top of the Stable-Baslines3 Contrib codebase's implementation of Quantile Regression DQN (QR-DQN). We choose to multiply the QR-DQN component of the CQL loss by 0.2. Using the notation proposed by Kumar et al. (2020), this is equivalent to $\alpha = 5$, which said work also uses. Other hyperparameters are $\tau = 1$, $\gamma = 0.99$, fixed learning rate of $1e - 4$, 100 epochs, and 50 quantiles (no exploration hyperparameters are specified, since we do not generate any new online data).

The policy architecture used for Habitat experiments are the same as those used for PPO, though the final network outputs quantile Q-values for each action (rather than just a distribution over actions). The action with the highest mean quantile value is chosen at evaluation time.

During training, we shuffle the data and load full offline trajectories until the buffer has at least $32 \times 1024 = 32768$ transitions or all trajectories have been loaded once that epoch. We then uniformly sample and train on batches of size 512 transitions from the buffer until each transition has been trained on once in expectation (e.g., $\sim \frac{\text{number of transitions in the buffer}}{512}$ batches). Each batch is used for 8 gradient steps before the next is sampled. We choose this data loading scheme to fit the training infrastructure provided by Stable-Baselines3 while not using up too much memory at once.

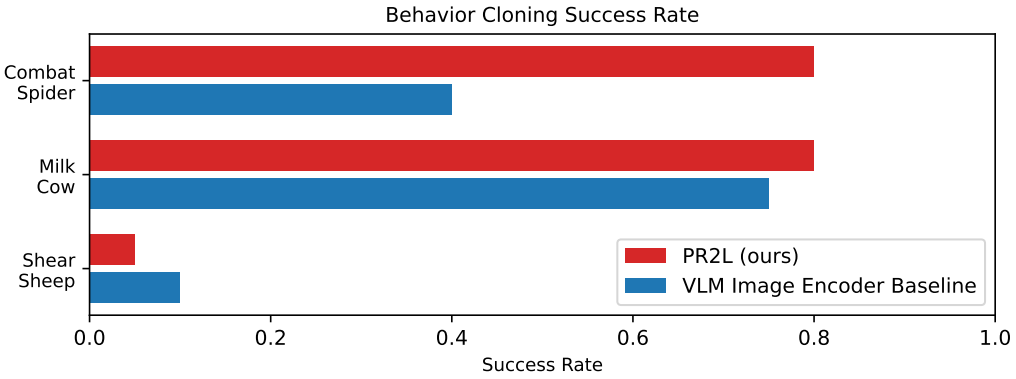

Figure 7. Success rates for BC on either PR2L or VLM image encoder baseline representations for all original tasks. PR2L excels at *combat spider*, even after the policy is trained for a single epoch.

D. Behavior Cloning Experiments

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 Figure 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 Figure 4). 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 E.

E. 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.

E.1. Minecraft Analysis

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 D). 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 Figure 8 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 policy. Additionally, we enlarge points corresponding to when the agent received rewards in order to recognize which actions aided in or achieved the task objective.

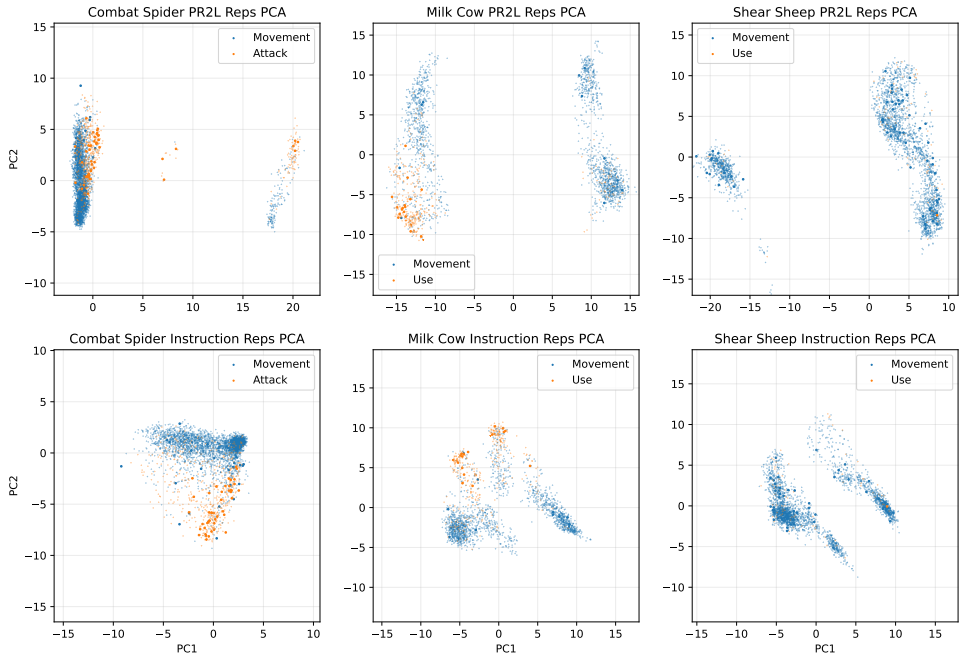

Figure 8. **PCA of PR2L representations of observations from twenty episode rollouts of expert policies in all three Minecraft tasks.** Larger points correspond to transitions where the expert received > 0.1 reward. We vary the prompt to be either our task-relevant prompt or the RT-2-style baseline instruction prompt. Our prompt's representations are bi-modal, with the clusters on the left corresponding to the VLM outputting "yes" (the entity is in view). We find that most functional actions (orange points) that yielded rewards are located in said clusters. Note that, since these expert policies are trained on top of MineCLIP embeddings, the *shear sheep* policy is not very performant, as seen in Figure 4.

We find that our considered prompts resulted in a bimodal distribution over representations, wherein the left-side cluster corresponds to the VLM outputting "yes (the entity is in view)" and the right-side one corresponds to "no." Additionally, observations resulting in functional actions that received rewards (large orange points in Figure 8) tend to be on the left-side ("yes") cluster for representations elicited by our prompt, but are more widely distributed in the instruction prompt case, in agreement with intuition. This is especially clear in the *milk cow* plot, wherein nearly all rewarding functional actions (using the bucket on the cow to successfully collect milk) are in the lower left corner.

This analysis supports that the representations yielded by InstructBLIP in response to our chosen style of prompts are more structured than representations from instructions. Such structure is useful in identifying and learning rewarding actions, even when said actions were taken from an expert policy trained on unrelated embeddings. This suggests that such representations may similarly be more conducive to being mapped to good actions via RL, which we observe empirically (as our prompt's representations yield more performant policies than the instructions for the RT-2-style baseline).

E.2. Habitat Analysis

Likewise, we conduct a similar analysis on the Habitat data. Specifically, we wish to see if PR2L produces representations that are conducive to extracting the *value function* of a good policy. Since the chosen Habitat ObjectNav prompt is "What room is this?" we expect the state representations to be clustered based on room categories. Intuitively, states corresponding to the room one is likely to find the target object should have the highest values.

As shown in Figure 9, we thus used PCA to project expert trajectories' PR2L and general image encoder state representations (generated with Habitat's geodesic shortest path follower) to two dimensions, then colored each one based on their value under said near-optimal policy. We also plotted the mean and standard deviation of all points labeled as each room,

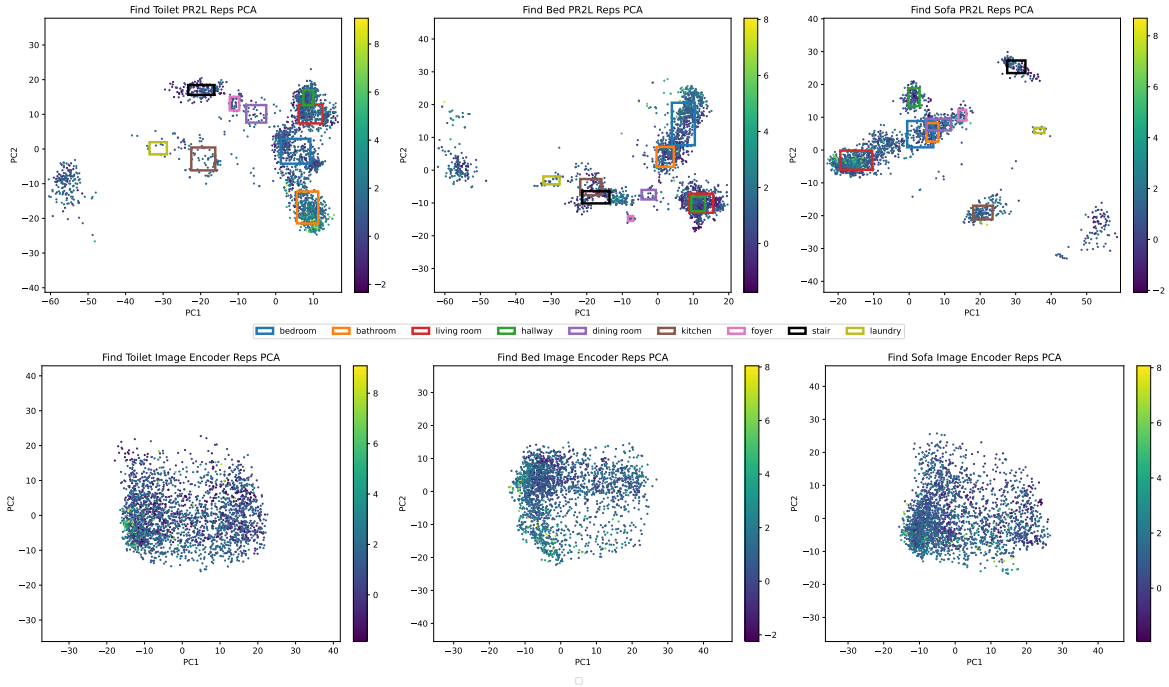

Figure 9. **PCA of PR2L (above) and image encoder (below) representations of observations from thirty episode rollouts of expert policies in all Habitat tasks.** The points' colors correspond to their value under Habitat's built-in oracle shortest path follower (a near-optimal policy). More yellow is better. Boxes correspond to points the VLM has labeled as a given household room, in response to the task prompt of "What room is this?" This analysis aligns with intuition: for *find toilet*, high value observations tend to be labeled as bathrooms (orange box), *find bed*'s tend to be labeled as bedrooms (blue), and *find sofa*'s are labeled as living rooms (red).

visualizing them as axis-aligned bounding boxes. Note that each upper subplot in Figure 9 has a cluster of points far from all boxes. These correspond to the VLM generating nothing or garbage data with no room label.

This visualization qualitatively agrees with intuition. High value states tend to be grouped with the room the corresponding target object is often found in: *find toilet* corresponds to bathrooms, *find bed* to bedrooms, and *find sofa* to living rooms. Comparatively, the general image encoder features do not have such semantically meaningful groupings; all observations are clustered together and, within that single grouping, high-value observations are more spread out. This all supports the idea that prompting allows representations to take on structures that correlate well to value functions of good policies.

F. Extended Literature Review

Learning in Minecraft. We now consider some current approaches for creating autonomous learning systems for tasks in Minecraft. Such works highlight some of the difficulties prevalent in tasks in said environment. For instance, since Minecraft tasks take place in a dynamic open world, it can be difficult for an agent to determine what goal it is attempting to reach and how close it is to reaching that goal. (Cai et al., 2023) tackles these issues by introducing and integrating a training scheme for self-supervised goal-conditioned representations and a horizon predictor. (Zhou et al., 2023) learns a model from visual observations to discriminate between expert state sequences and non-expert ones, which provides a source of intrinsic rewards for downstream RL tasks (as it pushes the policy to learn to match the expert state distribution, which tend to be "good" states for accomplishing tasks in Minecraft).

Foundation Models and Minecraft. Likewise, there has been much interest in applying foundation models – especially (V)LMs – to Minecraft tasks. (Baker et al., 2022) pretrains on large scale videos, which enabled the first agent that could learn to acquire diamond tools (thereby completing a longstanding challenge in the MineRL competition (Kanervisto et al., 2022)). LMs have subsequently also been used to produce graphs of proposed skills to learn or technology tree advancements to make in the form of structured language (Nottingham et al., 2023; Zhu et al., 2023; Yuan et al., 2023; Wang et al., 2023b). Other works propose to use the LLM to generate actions or code submodules given textual descriptions of observations or

agent states (Wang et al., 2023a). Finally, VLMs have been used largely for language-conditioned reward shaping (Fan et al., 2022; Ding et al., 2023). In contrast, we use VLMs as a source of representations for learning of atomic tasks (as defined by (Lin et al., 2023a)) that have pre-defined reward functions; the latter works can thus be used in conjunction with our proposed approach for tasks where these vision-language reward functions are appropriate.