Video PreTraining (VPT): Learning to Act by Watching Unlabeled Online Videos

Bowen Baker[*†] bowen@openai.com
Ilge Akkaya[*†] ilge@openai.com
Peter Zhokhov[*†] peterz@openai.com
Joost Huizinga[*†] joost@openai.com
Adrien Ecoffet[*†] adrien@openai.com
Brandon Houghton[*†] brandon@openai.com
Raul Sampedro[*†] raulsamg@gmail.com
Jie Tang[*†] jietang@openai.com
Jeff Clune[*†‡] jclune@gmail.com

Abstract

Pretraining on noisy, internet-scale datasets has been heavily studied as a technique for training models with broad, general capabilities for text, images, and other modalities.[1–6] However, for many sequential decision domains such as robotics, video games, and computer use, publicly available data does not contain the labels required to train behavioral priors in the same way. We extend the internet-scale pretraining paradigm to sequential decision domains through semi-supervised imitation learning wherein agents learn to act by watching online unlabeled videos. Specifically, we show that with a small amount of labeled data we can train an inverse dynamics model accurate enough to label a huge unlabeled source of online data – here, online videos of people playing Minecraft – from which we can then train a general behavioral prior. Despite using the native human interface (mouse and keyboard at 20Hz), we show that this behavioral prior has nontrivial zero-shot capabilities and that it can be fine-tuned, with both imitation learning and reinforcement learning, to hard-exploration tasks that are impossible to learn from scratch via reinforcement learning. For many tasks our models exhibit human-level performance, and we are the first to report computer agents that can craft diamond tools, which can take proficient humans upwards of 20 minutes (24,000 environment actions) of gameplay to accomplish.

1 Introduction

Work in recent years has demonstrated the efficacy of pretraining large and general found
models[7] on noisy internet-scale datasets for use in downstream tasks in natural language
computer vision.[5,6,8] For sequential decision domains (e.g. robotics, game playing, and c
usage) where agents must repeatedly act within an environment, a wealth of data also ex
web; however, most of this data is in the form of *unlabeled* video (i.e. without the ac
at each frame), making it much less straightforward to train a behavioral prior in th
than it is in e.g. natural language. In a few rare settings, such as Chess, Go, and S

[*]This was a large effort by a dedicated team. Each author made huge contributions on m
time periods. All members were full time on the project for over six months. BB, IA, PZ
original VPT project team and were thus involved for even longer (over a year). Aside fr
members, author order is random. It was also randomized between IA and PZ.

[†]OpenAI
[‡]University of British Columbia

already exist large datasets with action labels from various o[...] do[...] used for imitation learning.[9,10] When large labeled datasets do[...] for training capable agents is reinforcement learning (RL),[11] which c[...] expensive for hard-exploration problems.[12–18] Many virtual tasks, e.g. nav[...] Photoshop, booking flights, etc., can be very hard to learn with RL and do not ha[...] available sources of labeled data.[19,20] In this paper, we seek to extend the paradig[...] large, general-purpose foundation models to sequential decision domains by utilizing freely[...] internet-scale unlabeled video datasets with a simple semi-supervised imitation learning metho[...] call this method Video PreTraining (VPT) and demonstrate its efficacy in the domain of Minecraft.

Existing semi-supervised imitation learning methods aim to learn with few or no explicit action labels; however, they generally rely on the policy's ability to explore the environment throughout training, making them susceptible to exploration bottlenecks.[21–25] Furthermore, most prior semi-supervised imitation learning work was tested in the relatively low data regime; because we experiment with *far* more data ($\sim$70k hours of unlabeled video), we hypothesize that we can achieve good performance with a much simpler method, a trend that has proven true for pretraining in other modalities such as text.[1] In particular, given a large but unlabeled dataset, we propose generating pseudo-labels by gathering a small amount of labeled data to train an inverse dynamics model (IDM) that predicts the action taken at each timestep in a video. Behavioral cloning (BC) can require a large amount of data because the model must learn to infer intent and the distribution over future behaviors from only past observations. In contrast, the inverse dynamics modeling task is simpler because it is *non-causal*, meaning it can look at both past and future frames to infer actions. In most settings, environment mechanics are far simpler than the breadth of human behavior that can take place within the environment, suggesting that non-causal IDMs could require far less data to train than causal BC models. Using pseudo-labels generated from the IDM, we then train a model to mimic the distribution of behavior in the previously unlabeled dataset with standard behavioral cloning at scale, which does not require any model rollouts and thus does not suffer from any potential exploration bottlenecks in the environment. Finally, we show we can fine-tune this model to downstream tasks with either behavioral cloning or reinforcement learning.

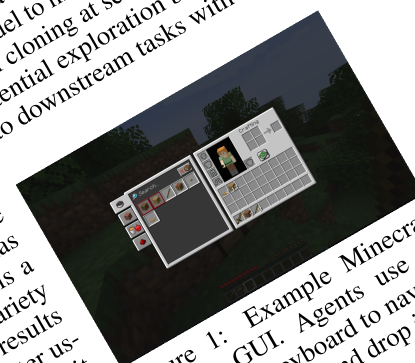


Figure 1: Example Minecraft crafting GUI. Agents use the mouse and keyboard to navigate menus and drag and drop items.

We chose to test our method in Minecraft because (a) it is one of the most actively played games in the world[26] and thus has a wealth of commonly available video data online, (b) it is a fairly open-ended sandbox game with an extremely wide variety of potential things to do, build, and collect, making our results more applicable to real-world applications such as computer usage, which also tends to be varied and open-ended, and (c) it has already garnered interest by the RL community as a research domain due to its complexity and correspondingly difficult exploration challenges.[27–31] In this work we use the native human interface for Minecraft so that we can (1) most accurately model the human behavior distribution and reduce domain shift between video data and the environment, (2) make data collection easier by allowing our human contractors to play the game without modification, and (3) eliminate the need to hand-engineer a custom interface for models to interact with the environment. This choice means that our models play at 20 frame[...] per second and must use a mouse and keyboard interface to interact with human GUIs for craftin[...] smelting, trading, etc., including dragging items to specific slots or navigating the recipe book w[...] the mouse cursor (Fig. 1). Compared to prior work in Minecraft that uses a lower frame rat[...] constructs crafting and attacking macros,[30,32–34] using the native human interface drastically in[...] the environment's exploration difficulty, making most simple tasks near impossible with R[...] scratch. Even the simple task of gathering a single wooden log while already facing a tre[...] consecutive attack actions with the human interface, meaning the chance for a naive rando[...] succeed is $\frac{1}{2}^{60}$. While this paper shows results in Minecraft only, the VPT method i[...] could be applied to any domain.

In Section 4 we show that the VPT foundation model has nontrivial zero-shot per[...] plishing tasks impossible to learn with RL alone, such as crafting planks and cr[...] requiring a human proficient in Minecraft a median of 50 seconds or $\sim$970 [...] Through fine-tuning with behavioral cloning to smaller datasets that target m[...] distributions, our agent is able to push even further into the technology t[...]

(taking a human a median of 2.3 minutes or ~2790 actions).
the most dramatic improvements: our agent is able to craft diamo
in Minecraft made even more challenging by using the native human
a proficient human a median upwards of 20 minutes or ~24000 actions.
of this work are (1) we are the first to show promising results applying semi-sup
learning to extremely large, noisy, and freely available video datasets for sequential dec
(2) we show that such pretraining plus fine-tuning enables agents to solve tasks that were
impossible to learn, (3) we show that labeled contractor data is far more efficiently used w
the VPT method than it would be by directly training a foundation model from it and (4) we open source our contractor data, trained model weights, and Minecraft environment for future research into learning to act via semi-supervised imitation learning at scale.

2 Preliminaries and Related Work

Imitation learning methods[35–38] seek to construct a policy that accurately models the distribution of behavior in some dataset $D = \{(o_i, a_i)\}, i \in \{1...N\}$ of action-observation pairs. In order to roll out these policies in an environment, they must be *causal*, meaning they condition on observations from the current timestep t and past timesteps only, i.e. $\pi \sim p(a_t|o_1...o_t)$. Imitation learning is simplest when demonstrations are labeled with corresponding actions. Imitating labeled trajectories has seen success in aerial vehicles,[39,40] self-driving cars,[41,42] board games,[9,43] and video games.[10,44]

When labeled demonstrations are not available, standard behavioral cloning will not work; however, there is a large body of work in imitating behavior from unlabeled demonstrations.[22] For instance, GAIL[23] constructs an adversarial objective incentivizing the trained policy to exhibit behaviors indistinguishable from those in the target dataset. Edwards et al.[45] propose to first learn a latent policy using unlabeled demonstrations and then map the learned latent actions to real actions with a small amount of environment interaction. Peng et al.[46] first use motion-capture methods to track agent positions in videos and then train RL agents to match these waypoints. Similarly, Behbahani et al.[47] and Aytar et al.[48] task a RL agent to match waypoints; however, they construct waypoints that are embeddings from unsupervised feature learning models. Pathak et al.[49] and Nair et al.[50] train goal conditioned policies to take actions that advance the current state towards expert-provided goal states expressed as high dimensional visual waypoints. Most similar to our own work, Torabi et al.[24] simultaneously train (1) an inverse dynamics model (IDM),[51] which aims to uncover the underlying action between timesteps given observations of past and future timesteps, e.g. $p_{\text{IDM}}(a_t|o_t, o_{t+1})$, and (2) a behavioral cloning (BC) model on trajectories of observations labeled with the IDM. Data to train the IDM is collected by rolling out the BC model in the target environment such that both models improve in tandem. However, at any point in training if there are sequences in the dataset that
the IDM performs poorly on, it requires that the BC model perform those or similar sequences in
order for the IDM to improve and correctly label them. Therefore, if the BC model does not explor
efficiently, it could severely slow down learning. In order to avoid this potential issue we opted fo
simpler two-stage approach: we first train an IDM on a small number of labeled trajectories colle
from human contractors (they play the game as would normally as we record their keypresse
mouse movements). Because human contractors reach most relevant parts of the state space,
hold the IDM fixed throughout BC training.

Compared to most previous work in semi-supervised imitation learning, we experiment i
more complex and open-ended environment of Minecraft. Minecraft is a voxel-base
game that, due its popularity and wide variety of mechanics, has attracted a vast a
research.[27,28,30–34,52–60] A large body of work focuses on small, custom-made M
with tasks such as navigation,[53,60] block placing,[54,55] instruction following,[58,5
others.[28,31,57] Work operating in the massive, randomly generated environments
has included hill climbing,[52] automated curriculum learning[30] and, most close
experiments presented in Sec. 4.4, diamond mining.[27,32–34] However, to the b
there is no published work that operates in the full, unmodified human actio
drag-and-drop inventory management and item crafting.

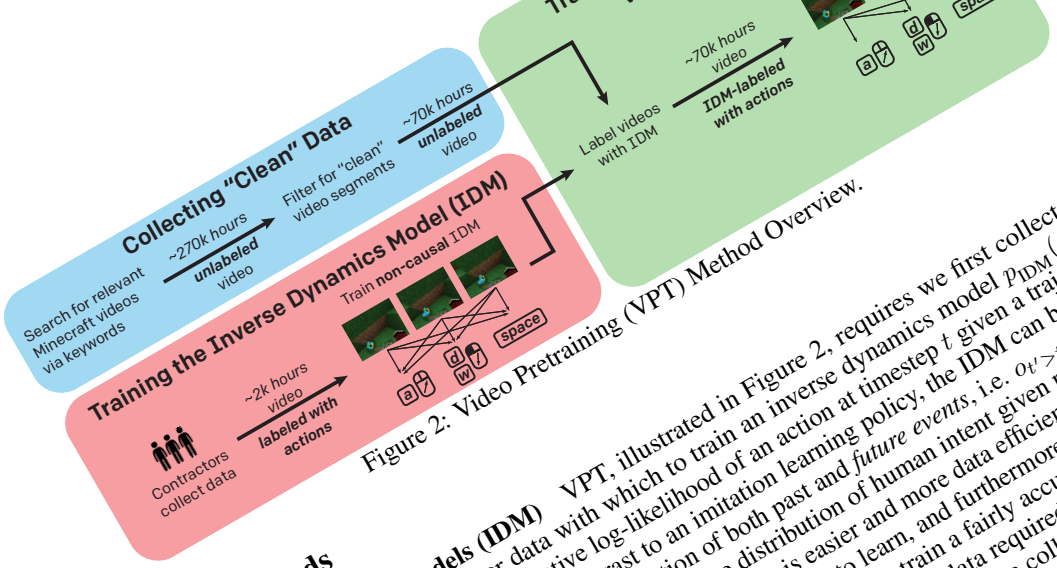


Figure 2: Video Pretraining (VPT) Method Overview.

3 Methods

Inverse Dynamics Models (IDM) VPT, illustrated in Figure 2, requires we first collect a small amount of labeled contractor data with which to train an inverse dynamics model $p_{\text{IDM}}(a_t|o_{1...T})$, which seeks to minimize the negative log-likelihood of an action at timestep t given a trajectory of observations $o_t : t \in [1...T]$. In contrast to an imitation learning policy, the IDM can be non-causal, meaning its prediction for a_t can be a function of both past and *future events*, i.e. $o_{t'>t}$. Compared to the behavioral cloning objective of modeling the distribution of human intent given past frames only, we hypothesize that inverting environment dynamics is easier and more data efficient to learn. Indeed, Sec. 4.1 will show that the IDM objective is much easier to learn, and furthermore Sec. 4.6 will show that with very little labeled data (as few as 100 hours) we can train a fairly accurate IDM. This IDM can be used to label online videos, providing the large amount of data required for the harder task of behavioral cloning. See appendices D and B for IDM training and data collection details.

Data Filtering We gather a large dataset of Minecraft videos by searching the web for related keywords (Appendix A). Online videos often (1) include overlaid artifacts, such as a video feed of the player's face, channel logos, watermarks, etc., (2) are collected from platforms other than a computer with different gameplay, or (3) are from different game modes, e.g. in Minecraft we only want "survival mode" where players start from scratch and must gather or craft all their items. We call data "clean" if it does not contain visual artifacts and is from survival mode, and call all other data "unclean." With enough data, a large enough model, and enough training compute, a BC model trained on both unclean and clean videos would likely still perform well in a clean Minecraft environment. However, for simplicity and training compute efficiency, we choose to filter out unclean segments of video (note that a video may contain both clean and unclean segments). We do this by training a model to filter out unclean segments using a small dataset (8800) of images sampled from online videos labeled by contractors as clean or unclean (Appendix A.2).

VPT Foundation Model We train a foundation model with standard behavioral cloning, i.e. minimizing the negative log-likelihood of actions predicted by the IDM on clean data. For a particular trajectory of length T we minimize

$$\min_{\theta} \sum_{t \in [1...T]} -\log \pi_\theta(a_t|o_1, \ldots, o_t), \text{ where } a_t \sim p_{\text{IDM}}(a_t|o_1, \ldots, o_t, \ldots, o_T)$$

As we will see in the following sections, this model exhibits nontrivial zero-shot behavior and fine-tuned with both imitation learning and RL to perform even more complex skills.

4 Results

4.1 Performance of the Inverse Dynamics Model

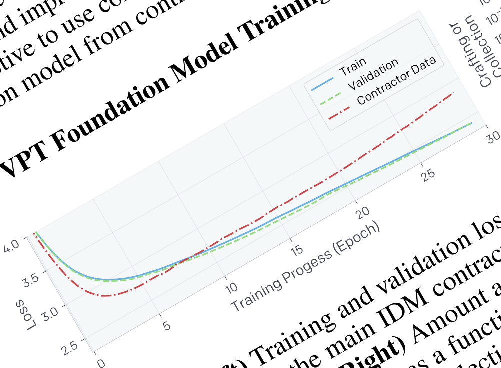
The IDM architecture is comprised primarily of a temporal convolution layer, processing stack, and residual unmasked attention layers, from which the I predicts keypresses and mouse movements (see Appendix D for IDM arc details). A key hypothesis behind our work is that IDMs can be trained with a of labeled data. While more data improves both mouse movement and keyp

Figure 3: **(Left)** IDM keypress accuracy and mouse movement R^2 (explained variance
function of dataset size. **(Right)** IDM vs. behavioral cloning data efficiency.

IDM trains on only 1962 hours of data (compared to the ~70k hours of clean data we collected from the internet) and achieves 90.6% keypress accuracy and a 0.97 R^2 for mouse movements evaluated on a held-out validation set of contractor-labeled data (Figure 3 left).

Figure 3 (right) validates our hypothesis that IDMs are far more data efficient than BC models, likely because inverting environment mechanics is far easier than modeling the entire distribution of human behavior. The IDM is two orders of magnitude more data efficient than a BC model trained on the same data and improves more quickly with more data. This evidence supports the hypothesis that it is more effective to use contractor data within the VPT pipeline by training an IDM than it is to train a foundation model from contractor data directly (Sections 4.5 and 4.6 provide additional evidence).

4.2 VPT Foundation Model Training and Zero-Shot Performance



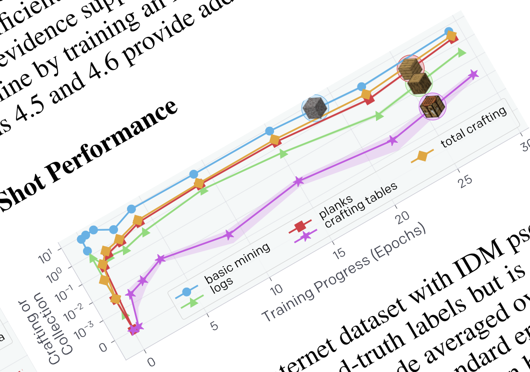


Figure 4: **(Left)** Training and validation loss on the web_clean internet dataset with IDM pseudo-labels, and loss on the main IDM contractor dataset, which has ground-truth labels but is out-of-distribution (see text). **(Right)** Amount a given item was collected per episode averaged over 2500 60-minute survival episodes as a function of training epoch, shaded with the standard error of the mean. Basic mining refers to collection of dirt, gravel, or sand (all materials that can be gathered without tools). Logs are obtained by repeatedly hitting trees for three seconds, a difficult feat for an RL agent to achieve as we show in Sec. 4.4. Planks can be crafted from logs, and crafting tables crafted from planks. Crafting requires using in-game crafting GUIs, and proficient humans take a median of 50 seconds (970 consecutive actions) to make a crafting table.

We now explore the emergent behavior learned by a behavioral cloning policy trained on an extremely
large, but noisy, internet dataset labeled with our IDM. To collect the unlabeled internet dataset
we searched for publicly available videos of Minecraft play with search terms such as "minecra
survival for beginners." These searches resulted in ~270k hours of video, which we filtered dow
"clean" video segments yielding an *unlabeled* dataset of ~70k hours, which we refer to as web_c
(Appendix A has further details on data scraping and filtering). We then generated pseudo
for web_clean with our best IDM (Section 3) and then trained the VPT foundation mo
behavioral cloning. Preliminary experiments suggested that our model could benefit from
of training and that a 0.5 billion parameter model was required to stay in the efficie
regime[63] for that training duration (Appendix H), which took ~9 days on 720 V100 G

We evaluate our models by measuring validation loss (Fig. 4, left) and rolling
Minecraft environment. Unless otherwise noted, in all environment evaluations we
standard survival mode game where they play for 60 minutes, i.e. 72000 consecut
plot the mean and shade the standard error of the mean for various game statis
and collection rates (Fig. 4, right). The VPT foundation model quickly lear
to collect logs, a task we found near impossible for an RL agent to achieve
interface (Sec. 4.4). It also learns to craft those logs into wooden planks a

to craft a crafting table, which are required to unlock most diamond tools. Human proficient in Minecraft approximately 50 seconds (970 consecutive game ticks) to these behaviors are fairly complex in the native human action space, the these items at a rate far below that of our proficient contractors, e.g. on average 5.44 crafting tables in 60 minutes of play versus 0.19 for the foundation model. a non-negligible amount of wooden sticks, which are required to make wooden tools; flowers and crafts dyes from them; kills zombies that appear during the night; hunts wild to collect various rare and mushrooms and eats them; and finds game-generated villages from which and pillar jump, which involves the agent repeatedly jumping and quickly placing a block below itself such that it climbs upward by making a pillar.[iv]

While training and validation loss decrease healthily over training (Fig. 4, left), loss on our contractor dataset (which the VPT model does not train on) begins increasing after 7 epochs. Contractor data could be out-of-distribution because our contractors may have a different distribution of play or because there is some impactful visual domain shift compared to videos from the web. While one could have expected this would be predictive of declining evaluation performance, we do not see notable game statistics from the VPT foundation model rollouts (Figure 4, right) decrease over training, and in the next section we show that BC fine-tuning performance continually improves as the VPT foundation model trains. We provide more insight into this curious phenomenon in Appendix H.

4.3 Fine-Tuning with Behavioral Cloning

Foundation models are designed to have a broad behavior profile and be generally capable across a wide variety of tasks. To incorporate new knowledge or allow them to specialize on a narrower task distribution, it is common practice to fine-tune these models to smaller, more specific datasets. [1] The VPT foundation model trained on the broad `web_clean` dataset had nontrivial zero-shot performance; it was able to craft a crafting table yet unable to go past this in the technology tree. As a case study into BC fine-tuning, we attempt to improve the VPT foundation model's ability to collect and craft these "early game" items by fine-tuning to two narrower datasets targeted at Minecraft behavior within the first few minutes of players starting in a fresh world. In the first dataset, `contractor_house`, contractors have 10 minutes to build a basic house from scratch using primarily wood, sand, and dirt. Collecting contractor data can be difficult and expensive, so we also construct a dataset `earlygame_keyword` by searching for videos online with descriptions that match keywords such as "new world", "let's play episode 1", etc.; this is a subset of `web_clean` and is labeled with the IDM. See Appendix B.4 and A.3 for full descriptions of both datasets.

Figure 5: **(Left)** Collection and crafting rates for three policies: the zero-shot VPT foundation model, and the VPT foundation model BC fine-tuned to the `earlygame_keyword` and the `contractor_house` datasets. BC fine-tuning to either dataset improves performance, including a median of 1.2 minutes (1390 actions) to construct wooden tools and 2.3 minutes (2790 actions) to construct stone tools. Proficient Minecraft players can construct wooden and stone tools. Proficient Minecraft players can construct wooden tools and 2.3 minutes throughout training *after* they are BC fine-tuned to the `contractor_house` dataset. **(Right)** Collection and crafting rates for VPT foundation model throughout foundation model training. Fig. crafting-related behaviors increase throughout foundation model training. Fig. task terms (logs, planks, crafting tables, and total crafting).

[iv] Sample videos: https://www.youtube.com/playlist?list=PLNAOIb_agjf3U3rSv

Fine-tuning to `early_game_keyword` results in a large boost
model: 2.5x more crafting tables, 6.1x more planks, 4.3x more log
(Fig. 5). However, when fine-tuning to this dataset we did not see
only a refinement of existing skills. We saw an even bigger improvement
`contractor_house` dataset: 213x more crafting tables, 59x more wooden pla
and 59x more crafting over all. In addition, we saw the emergence of crafting
requires placing a crafting table on the ground, opening it to reveal a new crafting interface,
using it to craft wooden tools. This entire sequence takes a proficient human player a median of
minutes (1390 consecutive actions) to accomplish. The model goes further and collects cobblestone,
which requires a wooden pickaxe to mine, and crafts stone tools, requiring it to again use a crafting
table; this takes a proficient human player a median of 2.3 minutes (2790 consecutive actions). We
also saw this model more frequently raiding villages that randomly spawn in the game, hunting
animals for food, in addition to many behaviors we saw performed by the foundation model.[v]

Despite the foundation model's zero-shot rollout performance plateauing 1/3 into training (Fig. 4,
right), fine-tuning performance *does* continue to increase throughout foundation model training
(Fig. 5, right). Additionally, there is a stark difference in performance when training from scratch vs.
fine-tuning from the VPT foundation model (Fig. 5 right, comparing the left and rightmost points).

4.4 Fine-Tuning with Reinforcement Learning

Figure 6: Typical sequence of items for obtaining a diamond pickaxe. Below each item is the median time and number of actions contractors required to obtain that item and the percentage of contractors that got the item within 10 minutes. The median time to obtain a diamond pickaxe is unknown (except that it is $>$ 20m) because contractors obtained this item in less than 50% of 20-minute episodes.

To demonstrate the efficacy of RL fine-tuning, we chose the challenging goal of obtaining a diamond pickaxe within 10 minutes starting from a fresh Minecraft survival world. Doing so involves acquiring a sequence of difficult-to-obtain items that require complex skills like mining, inventory management, crafting with and without a crafting table, tool use, operating a furnace, and mining at the lowest depths, where many hazards like enemies and lava exist (Fig. 6). Adding to the difficulty, progress can be easily lost by dropping items, destroying items, or dying. Obtaining a diamond pickaxe more often than not takes a proficient human over 20 minutes (24,000 actions).

Agents are rewarded for each item obtained in the sequence, with lower rewards for items that have to be collected in bulk and higher rewards for items near the end of the sequence. Agents are optimized with the phasic policy gradient[64] RL algorithm for $\sim$1.3 million episodes (roughly 1.4×10^{10} frames). Episodes last for 10 minutes. See Appendix G.1 for reward function and RL training details. Due to computational constraints, RL experiments use a $\sim$ 248 million parameter VPT model (Appendix H).

A major problem when fine-tuning with RL is catastrophic forgetting[65,66] because previously learne skills can be lost before their value is realized. For instance, while our VPT foundation model ne exhibits the entire sequence of behaviors required to smelt iron zero-shot, it *did* train on examp players smelting with furnaces. It therefore may have some latent ability to smelt iron once the prerequisites to do so have been performed. To combat the catastrophic forgetting of late such that they can continually improve exploration throughout RL fine-tuning, we add an Kullback-Leibler (KL) divergence loss between the RL model and the frozen pretrained

Training from a randomly initialized policy fails to achieve almost *any* reward, unde hard an exploration challenge the diamond pickaxe task is for RL in the native hum (Fig. 7a). The model never learns to reliably collect logs, typically the first of many a diamond pickaxe (Fig. 7b). RL fine-tuning from the VPT foundation model better (Fig. 7a), learning everything up to mining iron ore and crafting furnaces this agent fails at smelting an iron ingot, the next item required to get further in

[v] Sample Videos: https://www.youtube.com/playlist?list=PLNAOIb_agjf2yDSs

Figure 7: RL Fine-tuning results. **(a)** RL from a randomly initialized model fails to get almost any reward, RL fine-tuning from the VPT foundation model performs substantially better with a reward near 13, and RL fine-tuning from the early-game model performs best with a reward of 25. When training the early-game model without a KL loss to the original policy (*No KL-loss*) progress stalls after 100,000 episodes, suggesting that the skills necessary to make further progress have been catastrophically forgotten. **(b)** RL from a randomly initialized model occasionally collects sticks by breaking leaves (an easy but inefficient method of getting sticks that does not require logs or planks) and never learns to reliably collect logs. **(c)** RL fine-tuning from the VPT Foundation model learns everything in the curriculum up to iron ore and making furnaces, but fails to learn to use the furnace to smelt iron ingots. **(d)** RL fine-tuning from the early-game model learns to obtain (at human-level) all items in the sequence towards a diamond pickaxe and crafts a diamond pickaxe in 2.5% of episodes.

because the zero-shot probability that the VPT foundation model smelts an iron ingot is too low, even when given the prerequisite materials.

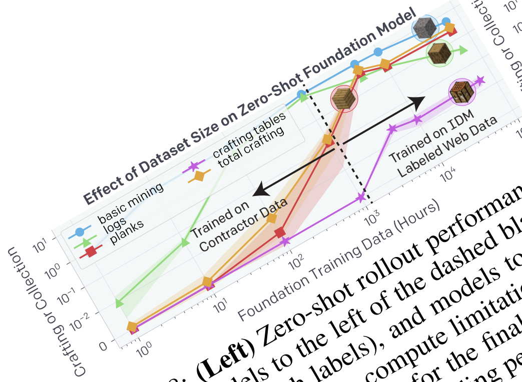
Results further improve by first BC fine-tuning the VPT Foundation Model to the `early_game_keyword` dataset (the *early-game model*, Sec. 4.3) and then fine-tuning with RL (Fig. 7a), which in preliminary experiments we found to perform better than first fine-tuning to `contractor_house` followed by fine-tuning with RL (Appendix G.2). The three-phase training (pretraining, BC fine-tuning, and then RL fine-tuning) succeeds in learning extremely difficult tasks: it achieves over 80% reliability on iron pickaxes, almost 20% reliability on collecting diamonds, and 2.5% reliability on obtaining a diamond pickaxe (Fig. 7d). For comparison, human players given the objective of obtaining a diamond pickaxe collect these items in 57%, 15%, and 12% of episodes, respectively, meaning our model is human-level for crafting iron pickaxes and mining diamonds. Others have managed to obtain diamonds with $\sim 0.1\%$ reliability in 15 minutes[32,33] but always with a simplified action space designed to ease exploration. To the best of our knowledge, **we are the first to report non-zero success rates on crafting a diamond pickaxe**. Qualitatively, the model developed useful skills for diamond mining, such as efficient mining patterns, cave exploration, returning to previously placed objects like crafting tables, and advanced techniques like using wooden pickaxes as fuel when moving on to iron tools.[vi]

Finally, we validated the importance of the KL loss to the pretrained model during RL fine-tuning. The treatment without a KL loss obtains only items early in the sequence (logs, planks, sticks, and crafting tables) limiting its reward (Fig. 7a). This failure to progress further into the sequence is likely because, while the initial skills of chopping logs and crafting planks are being learned with subsequent skills like crafting a wooden pickaxe are lost due to catastrophic forgetting.

4.5 Data Scaling Properties of the Foundation Model

In this section we validate a core hypothesis behind this work: that it is far more eff
labeled contractor data to train an IDM within the VPT method than it is to direc
foundation model from that same small contractor dataset. If we could cheaply c
contractor dataset of a similar order of magnitude as `web_clean`, then this would
however, collecting that scale of data would have cost millions of dollars.
foundation models trained on increasing orders of magnitude of data from 1 hou
`web_clean` dataset. Foundation models trained up to and including 1k hours

[vi] Videos found at https://www.youtube.com/playlist?list=PLNAOIb_agjf3e_UK



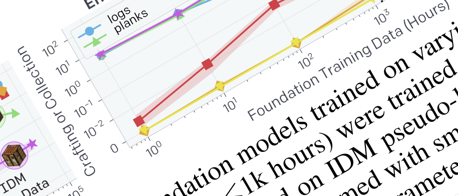


Figure 8: **(Left)** Zero-shot rollout performance of foundation models trained on varying am of data. Models to the left of the dashed black line (points ≤1k hours) were trained on contracto data (ground-truth labels), and models to the right were trained on IDM pseudo-labeled subsets of web_clean. Due to compute limitations, this analysis was performed with smaller (71 million parameter) models except for the final point, which is the 0.5 billion parameter VPT foundation model. **(Right)** The corresponding performance of each model *after* BC fine-tuning each model to the contractor_house dataset.

contractor data, and those trained on 5k hours and above are trained on subsets of web_clean, which does not contain any IDM contractor data. Scaling training data increases log collection, mining, and crafting capabilities. The zero-shot model only begins to start crafting crafting tables at over 5000 hours of training data. When fine-tuning each foundation model to contractor_house, we see that crafting rates for crafting tables and wooden tools increase by orders of magnitude when using the entire ~70k hour web_clean dataset. We furthermore only see the emergence of crafting stone tools at the largest data scale.

4.6 Effect of Inverse Dynamics Model Quality on Behavioral Cloning

This section investigates how downstream BC performance is affected by IDM quality. We train IDMs on increasingly larger datasets and use each to independently label the earlygame_keyword dataset (this smaller dataset was chosen due to a limited compute budget). We then train a BC model from scratch on each dataset and report game statistics for each model as a function of IDM contractor dataset size (Fig. 9).

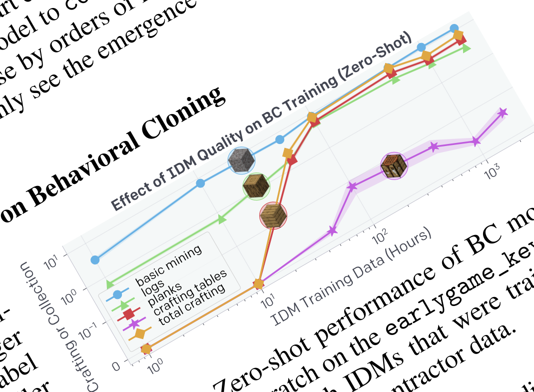


Figure 9: Zero-shot performance of BC models trained from scratch on the earlygame_keyword dataset labeled with IDMs that were trained on increasing amounts of contractor data.

IDMs trained on at least 10 hours of data are required for any crafting, and the crafting rate increases quickly up until 100 hours of data, after which there are few to no gains and differences are likely due to noise. Similarly, crafting tables are only crafted after 50 or more hours of IDM data, and again gains plateau after 100 hours. While in all previous experiments we use our best IDM trained on 1962 hours of data, these results suggest we could reduce that number to as low as 100 hours.

5 Discussion and Conclusion

The results presented in this paper help pave the path to utilizing the wealth of unlabeled data o web for sequential decision domains. Compared to generative video modeling or contrastive m that would only yield representational priors, VPT offers the exciting possibility of directly *to act* during pretraining and using these learned behavioral priors as extremely effective e priors for RL. VPT could even be a better general representation learning method eve downstream task is not learning to act in that domain—for example, fine-tuning to ex happening in a video—because arguably the most important information in any given present in features trained to correctly predict the distribution over future human a this intriguing direction to future work.

Future work could improve results with more data (we estimate we could col larger, better-tuned models. Furthermore, all the models in this work conditio only; we cannot ask the model to perform specific tasks. Appendix I presents on conditioning our models on closed captions (text transcripts of speech

become weakly steerable; we believe this a rich direction fo… consistently correlated with downstream evaluation metrics (Sec… slow and hard-won. Another fruitful future direction would be to invest… various training metrics and downstream evaluations. Finally, while we do … negative societal impacts from the models trained in this work, as VPT improves a… domains it will be important to assess and mitigate harms that emerge with other forms… on internet datasets, such as emulating inappropriate behavior.[67]

In conclusion, VPT extends the paradigm of training large and general purpose behavioral priors fr… freely available internet-scale data to sequential decision domains. Our models exhibited impressive zero-shot behavior and, when fine-tuned with RL, achieved an unprecedented result of crafting a diamond pickaxe in Minecraft (all the more difficult given the human interface). We further showed that contractor data is far better used within the VPT pipeline than to train a foundation model directly and that only a small amount of contractor data (about $2000 USD) was required to unlock massive amounts of unlabeled online data for use in BC. Finally, learning with the human keyboard and mouse interface is highly general and allows losslessly modeling the entire distribution of human behavior. While we only experiment in Minecraft, we believe that VPT provides a general recipe for training behavioral priors in hard, yet generic, action spaces in any domain that has a large amount of freely available unlabeled data, such as computer usage.

References

[1] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[3] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[4] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

[5] Dhruv Mahajan, Ross Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens Van Der Maaten. Exploring the limits of weakly supervised pretraining. In *Proceedings of the European conference on computer vision (ECCV)*, pages 181–196, 2018.

[6] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*… pages 8748–8763. PMLR, 2021.

[7] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney… Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. O… opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021…

[8] Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. Scaling vision… ers. *CoRR*, abs/2106.04560, 2021. URL `https://arxiv.org/abs/2106.0456`…

[9] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George V… che, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc La… tering the game of go with deep neural networks and tree search. *Nature*, 52… 2016.

[10] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu… young Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Geor… level in starcraft ii using multi-agent reinforcement learning. *Nature*, 5…

[11] Richard S Sutton and Andrew G Barto. *Reinforcement* 2018.

[12] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Pr Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et a scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[13] Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob M and Igor Mordatch. Emergent tool use from multi-agent autocurricula. *arXiv preprint arXiv:1909.07528*, 2019.

[14] Max Jaderberg, Wojciech M Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castaneda, Charles Beattie, Neil C Rabinowitz, Ari S Morcos, Avraham Ruderman, et al. Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, 2019.

[15] Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Zhaohan Daniel Guo, and Charles Blundell. Agent57: Outperforming the atari human benchmark. In *International Conference on Machine Learning*, pages 507–517. PMLR, 2020.

[16] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29, 2016.

[17] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

[18] Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. First return, then explore. *Nature*, 590(7847):580–586, 2021.

[19] Peter C Humphreys, David Raposo, Toby Pohlen, Gregory Thornton, Rachita Chhaparia, Alistair Muldal, Josh Abramson, Petko Georgiev, Alex Goldin, Adam Santoro, et al. A data-driven approach for learning to control computers. *arXiv preprint arXiv:2202.08137*, 2022.

[20] Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An open-domain platform for web-based agents. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3135–3144. PMLR, 06–11 Aug 2017. URL `https://proceedings.mlr.press/v70/shi17a.html`.

[21] Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.

[22] Faraz Torabi, Garrett Warnell, and Peter Stone. Recent advances in imitation learning from observation. *arXiv preprint arXiv:1905.13566*, 2019.

[23] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29, 2016.

[24] Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. *arXiv preprint arXiv:1805.01954*, 2018.

[25] YuXuan Liu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Imitation from observation: Learning to imitate behaviors from raw video via context translation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1118–1125. IEEE, 2018.

[26] Twinfinite Staff. Most played games in 2021, ranked by most players. *Twinfinite*. URL `https://twinfinite.net/2021/12/most-played-games-in-2020-ranked-by-peak-concurrent-players/`.

[27] William H Guss, Brandon Houghton, Nicholay Topin, Phillip Wang, Cayden Codel, Manuela Veloso, and Ruslan Salakhutdinov. Minerl: A large-scale dataset of minecraft demonstrations. *arXiv preprint arXiv:1907.13440*, 2019.

[28] Chen Tessler, Shahar Givony, Tom Zahavy, Daniel M
hierarchical approach to lifelong learning in minecraft. In *Pro*
on Artificial Intelligence, volume 31, 2017.

[29] Christian Scheller, Yanick Schraner, and Manfred Vogel. Sample ef
learning through learning from demonstrations in minecraft. In *NeurIPS 2019 C*
Demonstration Track, pages 67–76. PMLR, 2020.

[30] Ingmar Kanitscheider, Joost Huizinga, David Farhi, William Hebgen Guss, Brandon Hought
Raul Sampedro, Peter Zhokhov, Bowen Baker, Adrien Ecoffet, Jie Tang, et al. Multi-task
curriculum learning in a complex, visual, hard-exploration domain: Minecraft. *arXiv preprint*
arXiv:2106.14876, 2021.

[31] Junhyuk Oh, Valliappa Chockalingam, Honglak Lee, et al. Control of memory, active perception,
and action in minecraft. In *International Conference on Machine Learning*, pages 2790–2799.
PMLR, 2016.

[32] Vihang P Patil, Markus Hofmarcher, Marius-Constantin Dinu, Matthias Dorfer, Patrick M Blies,
Johannes Brandstetter, Jose A Arjona-Medina, and Sepp Hochreiter. Align-rudder: Learning
from few demonstrations by reward redistribution. *arXiv preprint arXiv:2009.14108*, 2020.

[33] Alexey Skrynnik, Aleksey Staroverov, Ermek Aitygulov, Kirill Aksenov, Vasilii Davydov, and
Aleksandr I Panov. Forgetful experience replay in hierarchical reinforcement learning from
demonstrations. *arXiv preprint arXiv:2006.09939*, 2020.

[34] Zichuan Lin, Junyou Li, Jianing Shi, Deheng Ye, Qiang Fu, and Wei Yang. Juewu-mc:
Playing minecraft with sample-efficient hierarchical reinforcement learning. *arXiv preprint*
arXiv:2112.04907, 2021.

[35] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. *Advances in*
neural information processing systems, 1, 1988.

[36] Stefan Schaal. Is imitation learning the route to humanoid robots? *Trends in cognitive sciences*,
3(6):233–242, 1999.

[37] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot
learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[38] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning:
A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):1–35, 2017.

[39] Claude Sammut, Scott Hurst, Dana Kedzier, and Donald Michie. Learning to fly. In *Machine*
Learning Proceedings 1992, pages 385–393. Elsevier, 1992.

[40] Alessandro Giusti, Jérôme Guzzi, Dan C Cireşan, Fang-Lin He, Juan P Rodríguez, Flavio
Fontana, Matthias Faessler, Christian Forster, Jürgen Schmidhuber, Gianni Di Caro, et al. A
machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotic*
and Automation Letters, 1(2):661–667, 2015.

[41] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Pr
Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end l
for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[42] Felipe Codevilla, Matthias Müller, Antonio López, Vladlen Koltun, and Alexey D
End-to-end driving via conditional imitation learning. In *2018 IEEE internation*
on robotics and automation (ICRA), pages 4693–4700. IEEE, 2018.

[43] Rémi Coulom. Computing "elo ratings" of move patterns in the game of go.
(4):198–208, 2007.

[44] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul,
John Quan, Andrew Sendonaris, Ian Osband, et al. Deep q-learning f
Proceedings of the AAAI Conference on Artificial Intelligence, volum

[45] Ashley Edwards, Himanshu Sahni, Yannick Schroecker, and C
policies from observation. In *International conference on ma*
PMLR, 2019.

[46] Xue Bin Peng, Angjoo Kanazawa, Jitendra Malik, Pieter Abbeel, and
Reinforcement learning of physical skills from videos. *ACM Transactions On G*
37(6):1–14, 2018.

[47] Feryal Behbahani, Kyriacos Shiarlis, Xi Chen, Vitaly Kurin, Sudhanshu Kasewa, Ciprian Stirbu,
Joao Gomes, Supratik Paul, Frans A Oliehoek, Joao Messias, et al. Learning from demonstration
in the wild. In *2019 International Conference on Robotics and Automation (ICRA)*, pages
775–781. IEEE, 2019.

[48] Yusuf Aytar, Tobias Pfaff, David Budden, Thomas Paine, Ziyu Wang, and Nando De Fre-
itas. Playing hard exploration games by watching youtube. *Advances in neural information*
processing systems, 31, 2018.

[49] Deepak Pathak, Parsa Mahmoudieh, Guanghao Luo, Pulkit Agrawal, Dian Chen, Yide Shentu,
Evan Shelhamer, Jitendra Malik, Alexei A. Efros, and Trevor Darrell. Zero-shot visual imitation.
In *ICLR*, 2018.

[50] Ashvin Nair, Dian Chen, Pulkit Agrawal, Phillip Isola, Pieter Abbeel, Jitendra Malik, and Sergey
Levine. Combining self-supervised learning and imitation for vision-based rope manipulation.
pages 2146–2153, 05 2017. doi: 10.1109/ICRA.2017.7989247.

[51] Duy Nguyen-Tuong, Jan Peters, Matthias Seeger, and Bernhard Schölkopf. Learning inverse
dynamics: a comparison. In *European symposium on artificial neural networks*, number CONF,
2008.

[52] David Abel, Alekh Agarwal, Fernando Diaz, Akshay Krishnamurthy, and Robert E Schapire.
Exploratory gradient boosting for reinforcement learning in complex domains. *arXiv preprint*
arXiv:1603.04119, 2016.

[53] Dilip Arumugam, Jun Ki Lee, Sophie Saskin, and Michael L Littman. Deep reinforcement
learning from policy-dependent human feedback. *arXiv preprint arXiv:1902.04257*, 2019.

[54] Alexander Trott, Stephan Zheng, Caiming Xiong, and Richard Socher. Keeping your dis-
tance: Solving sparse reward tasks using self-balancing shaped rewards. *Advances in Neural*
Information Processing Systems, 32, 2019.

[55] Stephan Alaniz. Deep reinforcement learning with model learning and monte carlo tree search
in minecraft. *arXiv preprint arXiv:1803.08456*, 2018.

[56] Hiroto Udagawa, Tarun Narasimhan, and Shim- Young Lee. Fighting zombies in minecraft with
deep reinforcement learning. Technical report, Technical report, Technical report, Stanford
University, 2016.

[57] Tianmin Shu, Caiming Xiong, and Richard Socher. Hierarchical and interpretable skill acq
tion in multi-task reinforcement learning. *arXiv preprint arXiv:1712.07294*, 2017.

[58] Junhyuk Oh, Satinder Singh, Honglak Lee, and Pushmeet Kohli. Zero-shot task gener
with multi-task deep reinforcement learning. In *International Conference on Machine*
pages 2661–2670. PMLR, 2017.

[59] Zhengxiang Shi, Yue Feng, and Aldo Lipani. Learning to execute or ask clarific
arXiv preprint arXiv:2204.08373, 2022.

[60] Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. Teacher-
learning. *IEEE transactions on neural networks and learning systems*, 31(

[61] Robert George Douglas Steel, James Hiram Torrie, et al. Principles and
Principles and procedures of statistics., 1960.

[62] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian S
recognition. In *Proceedings of the IEEE conference on comput*
pages 770–778, 2016.

[63] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin
Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for n
models. *arXiv preprint arXiv:2001.08361*, 2020.

[64] Karl W Cobbe, Jacob Hilton, Oleg Klimov, and John Schulman. Phasic policy gradient.
Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on*
Machine Learning, volume 139 of *Proceedings of Machine Learning Research*, pages 2020–
2027. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/cobbe21a.
html.

[65] Dhireesha Kudithipudi, Mario Aguilar-Simon, Jonathan Babb, Maxim Bazhenov, Douglas
Blackiston, Josh Bongard, Andrew P Brna, Suraj Chakravarthi Raja, Nick Cheney, Jeff Clune,
et al. Biological underpinnings for lifelong learning machines. *Nature Machine Intelligence*, 4
(3):196–210, 2022.

[66] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins,
Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al.
Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of*
sciences, 114(13):3521–3526, 2017.

[67] Emily M Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. On
the dangers of stochastic parrots: Can language models be too big???. In *Proceedings of the*
2021 ACM Conference on Fairness, Accountability, and Transparency, pages 610–623, 2021.

[68] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion,
Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-
learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830,
2011.

[69] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez,
Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information*
processing systems, 30, 2017.

[70] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint*
arXiv:1607.06450, 2016.

[71] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop.
In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[72] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint*
arXiv:1412.6980, 2014.

[73] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chana
Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andr
Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamk
Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative styl
performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F.
Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing S*
pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.
9015-pytorch-an-imperative-style-high-performance-deep-learni
pdf.

[74] Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Su
Yoshua Bengio. Identifying and attacking the saddle point problem in hig
convex optimization. *Advances in neural information processing systems*

[75] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How tra
deep neural networks? *Advances in neural information processing sy*

[76] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, [illegible]
Transformer-xl: Attentive language models beyond a fixe
arXiv:1901.02860, 2019.

[77] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Ole
policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[78] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel
dimensional continuous control using generalized advantage estimation. *arXiv prepri*
arXiv:1506.02438, 2015.

[79] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforce-
ment learning. *Machine learning*, 8(3):229–256, 1992.

[80] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap,
Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforce-
ment learning. In *International conference on machine learning*, pages 1928–1937. PMLR,
2016.

[81] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder,
Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience
replay. *Advances in neural information processing systems*, 30, 2017.

[82] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function ap-
proximators. In *International conference on machine learning*, pages 1312–1320. PMLR,
2015.

[83] Open Ended Learning Team, Adam Stooke, Anuj Mahajan, Catarina Barros, Charlie Deck,
Jakob Bauer, Jakub Sygnowski, Maja Trebacz, Max Jaderberg, Michael Mathieu, et al. Open-
ended learning leads to generally capable agents. *arXiv preprint arXiv:2107.12808*, 2021.

[84] Jelena Luketina, Nantas Nardelli, Gregory Farquhar, Jakob Foerster, Jacob Andreas, Edward
Grefenstette, Shimon Whiteson, and Tim Rocktäschel. A survey of reinforcement learn-
ing informed by natural language. In *Proceedings of the Twenty-Eighth International Joint*
Conference on Artificial Intelligence, IJCAI-19, pages 6309–6317. International Joint Confer-
ences on Artificial Intelligence Organization, 7 2019. doi: 10.24963/ijcai.2019/880. URL
https://doi.org/10.24963/ijcai.2019/880.

[85] DeepMind Interactive Agents Team, Josh Abramson, Arun Ahuja, Arthur Brussee, Federico
Carnevale, Mary Cassin, Felix Fischer, Petko Georgiev, Alex Goldin, Tim Harley, et al. Creating
multimodal interactive agents with imitation and self-supervised learning. *arXiv preprint*
arXiv:2112.03763, 2021.

[86] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical
text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.

[87] Dong Yu and Li Deng. *Automatic speech recognition*, volume 1. Springer, 2016.

[88] Daulet Nurmanbetov. rpunct, May 25 2021. URL https://github.com/Felflare/rpun
accessed 2022-04-22.

[89] Arvind Neelakantan, Tao Xu, Raul Puri, Alec Radford, Jesse Michael Han, Jerry T
Qiming Yuan, Nikolas Tezak, Jong Wook Kim, Chris Hallacy, et al. Text and code em
by contrastive pre-training. *arXiv preprint arXiv:2201.10005*, 2022.

Acknowledgements

We thank the following people for helpful discussions and support: Bob McG
Joel Lehman, Ilya Sutskever, Wojciech Zaremba, Ingmar Kanitscheider, David
Jonathan Gordon, and the OpenAI supercomputing team, especially Christia
and Christopher Berner.

Supplementary Inform

A Collecting Internet Data

A.1 Initial Unclean Dataset Curation

Our goal was to curate a video dataset of Minecraft gameplay from the survival game mode. A
ally, we prefer the data come from game modes as close as possible to our evaluation environme
meaning preferably coming from Minecraft version 1.16, being on a computer (which uses a mouse
and keyboard vs. video game controllers with keypads and other buttons), being single- (vs. multi-)
player, and having the default look of the game (vs. modifications that alter that style, such as to
make it look realistic). To try to accomplish these goals, we collect a dataset by performing keyword
searches of publicly available videos on the internet. A list of search queries we used are given in
Table 1.

minecraft survival longplay
minecraft gameplay no webcam
minecraft gameplay survival mode
minecraft survival tutorial
minecraft survival guide
minecraft survival let's play
minecraft survival for beginners
minecraft beginners guide
ultimate minecraft starter guide
minecraft survival guide 1.16
minecraft how to start a new survival world
minecraft survival fresh start
minecraft survival let's play episode 1
let's play minecraft episode 1
minecraft survival 101
minecraft survival learning to play
minecraft survival
how to play minecraft survival
how to play minecraft
minecraft survival basic
minecraft survival for noobs
minecraft survival for dummies
minecraft survival for beginners
how to play minecraft for beginners
minecraft survival tutorial series
minecraft survival new world
minecraft survival a new beginning
minecraft survival episodio 1
minecraft survival episode 1
minecraft survival 1. bölüm
i made a new minecraft survival world

Table 1: Search terms used for generating the initial web dataset.

For videos that have metadata available, we perform an additional step of metadata-based f
to eliminate videos that do not fit our target distribution. In this step, we look for a list of
keywords in the video title and description and reject videos that contain these terms. Th
keywords we use are: {ps3, ps4, ps5, xbox 360, playstation, timelapse, multiplayer, r
pocket edition, skyblock, realistic minecraft, how to install, how to download, realmcra
This process yielded us ∼270k hours of unlabeled data, which we filter down to only
as described in the next section.

A.2 Training a Model to Filter out Unclean Video Segments

We restrict the scope of this work to the Minecraft Survival game mode
training dataset to clips that are obtained from this mode that are relatively

To do so, we asked contractors to label a set of random video f
(N=8800). These images were from a random subset of the videos
of the project (Section A.1).

A.2.1 Label Collection

We asked 5 workers on Amazon Mechanical Turk (mTurk) that we selected with a sample qu
task to label random screen capture images to be used in training the classifier. A sample w
interface that the workers saw on mTurk is given in Figure 10.

We asked workers to label videos as being in one of the following three categories (see Figure 11 for visual examples of each class):

1. `Minecraft Survival Mode - No Artifacts`: Video frames (images) that correspond to the Minecraft Survival game mode that do not contain any non-game visual artifacts (e.g. subscribe buttons, channel logos, advertisements, picture-in-picture of the narrator, etc.).
2. `Minecraft Survival Mode - with Artifacts`: Video frames (images) of the Minecraft Survival game mode that include such visual artifacts.
3. `None of the Above`: Video frames (images) that are not from the Minecraft survival game mode, including those from other Minecraft game modes such as *creative* mode or even other games/topics entirely.

The full set of instructions workers received are as follows (note that we also included multiple image examples from each category in the worker instructions, similar to the sample subset provided in Figure 11):

Please help us identify screenshots that belong only to the survival mode in Minecraft. Everything else (Minecraft creative mode, other games, music videos, etc.) should be marked as `None of the Above`. Survival mode is identified by the info at the bottom of the screen:

- a health bar (row of hearts)
- a hunger bar (row of chicken drumsticks)
- a bar showing items held

Survival Mode
Valid survival mode videos have health/hunger bars and an item hotbar at the bottom of the screen.

Creative Mode
Creative mode only has an item hotbar and should be classified as `None of the Above`.

Label Descriptions

- `Minecraft Survival Mode - No Artifacts`: These images will be clean screenshots from the Minecraft survival mode gameplay without any noticeable artifacts.
- `Minecraft Survival Mode - with Artifacts`: These images will be valid survi mode screenshots, but with some added artifacts. Typical artifacts may include im overlays (a logo/brand), text annotations, a picture-in-picture of the player, etc.
- `None of the Above`: Use this category when the image is not a valid Minecraft screenshot. It may be a non-Minecraft frame or from a different game mode. In nor game modes such as the creative mode, the health/hunger bars will be missin image, the item hotbar may or may not be still present.

In total, we spent $319.96 on human labeling experiments on mTurk, of which $159.9 paid to workers. The remaining amount was spent towards Amazon platform fe received $0.01 per labeled image, at an hourly compensation of $7.20 (based on an time of 5 seconds/image – in our internal sample run of the same task, we found time to be < 3 seconds).

Since we perform rigorous keyword and metadata based filtering of videos (a which we served sample images to be labeled, serving offensive content t

low risk and no such images were detected during our ma
during our experiment, and the workers were fully anonymized via
personally identifiable information (PII) was collected.

Figure 10: Amazon Mechanical Turk worker interface showing an example labeling task

Figure 11: **(Left)** Sample image for Class 1: `Minecraft Survival Mode - No Artifacts`. **(Middle)** Sample image for Class 2: `Minecraft Survival Mode - with Artifacts` – Image contains annotations and picture-in-picture of the narrator. **(Right)** Sample image for Class 3: `None of the Above` – Image is missing the hotbar as well as health and armor bars, indicating that it was not captured during survival mode gameplay

A.2.2 SVM Training

With the image labels collected as described in the previous section, we trained a classifier to extract video segments that consist of frames from the `Minecraft Survival Mode - No Artifacts` category. Given a set of labeled images, we obtain embeddings for each image using the RN50x64 ResNet CLIP Model.[6] This is a ResNet-based CLIP model that is scaled up to have approximately 64x the compute of a ResNet-50. We then train a Support Vector Machine (SVM) using the RP kernel to obtain a frame classifier. We use the Scikit-learn[68] SVM implementation with the param configuration given in Table 2.

Finally, we apply the classifier to frames of raw video sequences at a rate of 3 frames/second. W for videos that consist of at least 80% "clean" frames at this stage (Classes `Minecraft S Mode - with Artifacts` and `None of the Above` are both considered not clean). Fr we apply a median filter (with a kernel size of 7) to the labels and segment videos by "clean" segments that are at least 5s in duration. The result of this is our final `web_cl`

A.3 `early_game` Dataset

The `early_game` dataset is a ∼3000 hour subset of `web_clean` targeted at "ea behavior, i.e. instances where players start in a fresh world with no items. W text that accompanies the videos in `web_clean` and determine whether any expressions match:

CLIP Model Specification	Kr[illegible]	
CLIP Input Image Resolution	448x44[illegible]	
CLIP Embedding Feature Length	1024	
SVM Parameters	Kernel	rbf
	C	20
	Gamma	scale
Sample Size	Class 1	2200
	Class 2	2200
	Class 3	4400

Table 2: Feature Extraction Details and SVM Configuration. The parameters are for the SVM implementation in Scikit-learn[68].

- (ep|episode|eps|day|session|sesh|chapter|chap .|series|part|parte|pt|round|day|tập|bölüm|episodio|епизод|эпизод)( )*(\.1|#1|1|\.01|#01|01|one[^0-9]|$)
- start
- beginning
- (new|fresh|clean).*(world|game|play)
- from scratch

From this set of videos, we take only the first 5 minutes of each video.

B Contractor Data

B.1 Recording Contractor Play

Our contractors use a custom Minecraft recorder that we built that records their actions and game video feeds as they play. The recorder is implemented using the MCP-Reborn (github.com/Hexeption/MCP-Reborn) modding package. To ensure that the recorder environment is as close as possible to the Minecraft environment used for RL rollouts and evaluations (Appendix C), we use the same underlying game engine for both. The recorder is a Java app that runs in a window mode, with constant resolution of 1280x760. Brightness is set to 0 (the "gloomy" setting in Minecraft), which is the default setting. Other graphics settings (field of view, GUI scale) are fixed to the values used in the Minecraft environment (C.1); we explicitly prevented users from changing graphics settings. Unlike the environment, the recorder allows all keyboard key presses and continuous (as opposed to binned) mouse actions. On every game step (or "tick") the frame buffer used to display the game window is downsized to 640x360 and written into a video file. In-game actions are recorded in a separate JSONL file (a text file where each line is a JSON-formatted string). All recordings are chunked into 5 minute clips: after each 5 minute segment of contractor game play the recorde[illegible] automatically uploads the video file, the JSONL file with actions, as well as a Minecraft state f[illegible] To ensure that contractors cannot corrupt each other's data, we provided every contractor wit[illegible] individual cloud bucket, as well as with credentials giving write access only to that bucket. Crede[illegible] also included adjective-adjective-noun names (e.g. grumpy-amethyst-chipmunk), generated w[illegible] `namegenerator` python package to ensure contractor anonymity when we publish the data[illegible]

B.2 Contractor Contract

We recruited contractors by posting the following offer on the UpWork freelancing [illegible]

"We are collecting data for training AI models in Minecraft. You'll need [illegible] java, download the modified version of Minecraft (that collects and u[illegible] play data), and play Minecraft survival mode! Paid per hour of gam[illegible] experience in Minecraft not necessary. We do not collect any data th[illegible] to Minecraft from your computer."

We had the applications open for a day, and then randomly selected contractors. Later in the project, as we needed more data and as some contractors left their contracts, we added more applicants from the original pool as well as new working contractors. The contractors were paid $20 per hour (minus Upwork applicable taxes). All of the results presented in this paper are based on about (including data recorded to gather statistics of human play that was not used for us around $90,000. Over the course of the project, we collected some data we did not bugs in the recorder and for some ideas we ultimately did not pursue. In total, we spent about for contractor compensation over the course of the project. However, as we discuss in Sec. 4.6, we could likely obtain most of our results with an IDM trained using only $2000 worth of data, i.e. the foundation VPT model, BC fine-tuning to the `early_game_keyword` dataset, and the RL fine-tuning results. Collecting the `contractor_house` dataset cost about $8000. Because we used the IDM trained on about 2000 hours of contractor data, the actual cost of contractor data for those results was around $40,000.

In early stages of the project, we were planning to use contractor data solely for the purpose of training the IDM. As such, no specific tasks were given, other than "play the survival mode of Minecraft like you normally would." Later in the project, we requested that contractors perform specific tasks in Minecraft, such as:

- Collect as many units of wood as possible, using only wooden or stone tools (`treechop`)
- Start a new world every 30 minutes of game play
- Build a basic house in 10 minutes using only dirt, wood, sand, and either wooden or stone tools (`contractor_house`, more details below in Appendix B.4).
- Starting from a new world and an empty inventory, find resources and craft a diamond pickaxe in 20 minutes (`obtain_diamond_pickaxe`). This dataset was used to obtain statistics for how long it takes humans on average to complete this task (and the subtasks required to complete it) when obtaining a diamond pickaxe is their goal.

Since we only recorded in-game events and videos, the data does not include personally identifiable information. That being said, the contractors could theoretically use Minecraft's open-world property to generate personally identifiable information and/or offensive content (e.g. by using Minecraft blocks to write their name or offensive messages, then finding a spot from which the message would be visible). In practice, we have not seen any attempts to do so in the contractor videos that we watched. Of course, we train our BC models on videos from the internet of people playing Minecraft, and if such behavior is in those videos our model could also potentially learn it, although we expect such behavior is rare enough that our model would not be likely to reproduce it.

B.3 Data for the Inverse Dynamics Model.

Since the IDM's task is to infer actions given the video, any labelled data is appropriate for IDM training. In practice, we included general gameplay as well as the `treechop` task data described in the previous section, which amounted to a total of 1962 hours. Due to collecting datasets like `contractor_house` only at late stages of the project, they were not included in IDM training.

B.4 contractor_house.

The `contractor_house` contains about 420 hours of data. We asked contractors to build a house in 10 minutes, using only basic dirt, wood, and sand, blocks. Each trajectory starts in a generated world and a timer forcibly ends a trajectory after a 20 minute time limit. For this task, contractors chose to begin their trajectories by crafting basic tools and building blocks, it was common for the first 2 minutes to be spent crafting a wooden pickaxe and then for an assortment of stone tools before gathering more building blocks and beginning structure.

C Minecraft environment details

Our Minecraft training environment is a hybrid between MineRL[27] (github.com/Hexeption/MCP-Reborn) Minecraft modding package. Unli

Figure 12: **(Left)** Sample of a Minecraft frame in the original resolution (640x360) with an in-game GUI open. The mouse cursor can be seen in the center of the image. This particular GUI shows the player's inventory and can be used to craft very basic items. **(Middle)** We downsample images to 128x128 for computational reasons. Shown is a downsampled observation with an in-game GUI for crafting. This is the resolution used by our models. **(Right)** A 128x128 observation as seen by our models without in-game GUI. The health, hunger, hotbar overlays, and agent hand can be seen in the lower part of the image.

game, in which the server (or the "world") always runs at 20Hz and the client runs as fast as rendering can complete (typically at 60-100Hz), in our version the client and server run in the same thread at the same frequency. This allows us to run the environment slower or faster than real time, while avoiding artifacts like missing chunks of the world. The action and observation spaces are similar to those of MineRL environments and are described in more detail in the following subsections. The environment also returns diagnostic information, such as in-game stats, contents of the agent's inventory, whether any in-game GUI is open, etc., which we use for tracking and recording but not as inputs to the models. The episode length is 10 minutes for RL experiments and 60 minutes for BC model evaluations. The agent can "die" in a number of ways, such as staying under water for too long and drowning, being killed by hostile mobs, or falling from a tall structure. We do not terminate the episode on agent "death". Instead, just as for humans in the regular Minecraft game, the agent drops all its items when it dies and respawns at a random spot close to the initial spawning spot in the same Minecraft world. The policy state is not masked on death, so the model can remember the fact that it has died and act accordingly.

C.1 Observation space

The environment observations are simply the raw pixels from the Minecraft game that a human would see. Unlike MineRL, we do not remove overlays like the hotbar, health indicators, and the animation of a moving hand shown in response to the attack or "use" actions. The field of view is 70 degrees, which corresponds to the Minecraft default. GUI scale (a parameter controlling the size of the in-game GUI) is set to 2, and brightness is set to 2 (which is not a Minecraft default, but is very frequently used in online videos). The rendering resolution is 640x360, which is downsampled to 128x128 before being input to the models. We empirically found 128x128 to be the smallest resolution for which in-game GUI elements are still discernible, and then chose that to minimize compute costs. Whenever an in-game GUI is open, we additionally render an image of a mouse cursor at the appropriate mouse position to match what a human player's operating system does (Fi 12).

C.2 Action space

Our action space includes almost all actions directly available to human players, such as k mouse movements, and clicks. The specific binary actions we include are shown in Tab

One difference between the human action space and our agent's is that we disallow ty letters, which is only useful for entering text into the search bar of the crafting recipe can either do that or browse the recipe book with the mouse, the latter of which c do. However, because we do allow the agent to press letters that are also shortcu outside of the GUI, the "W" key triggers the `forward` action) agents are able within the GUI (W, A, S, D, E, Q) that produce letters if the recipe book sear have not seen agents attempt to search the recipe book with these letters. Inst the recipe book with the mouse or craft by dragging items around the craft

Action	Human action	Description
forward	W key	Move forward.
back	S key	Move backward.
left	A key	Strafe left.
right	D key	Strafe right.
jump	space key	Jump.
inventory	E key	Open or close inventory and the 2x2 crafting grid.
sneak	shift key	Move carefully in current direction of motion. In the GUI it acts as a modifier key: when used with `attack` it moves item from/to the inventory to/from the hotbar, and when used with `craft` it crafts the maximum number of items possible instead of just 1.
sprint	ctrl key	Move fast in the current direction of motion.
attack	left mouse button	Attack; In GUI, pick up the stack of items or place the stack of items in a GUI cell; when used as a double click (attack - no attack - attack sequence), collect all items of the same kind present in inventory as a single stack.
use	right mouse button	Place the item currently held or use the block the player is looking at. In GUI, pick up the stack of items or place a single item from a stack held by mouse.
drop	Q key	Drop a single item from the stack of items the player is currently holding. If the player presses ctrl-Q then it drops the entire stack. In the GUI, the same thing happens except to the item the mouse is hovering over.
hotbar.[1-9]	keys 1 – 9	Switch active item to the one in a given hotbar cell.

Table 3: Binary actions included in the action space. `https://minecraft.fandom.com/wiki/Controls` has more detailed descriptions of each action.

In addition to the binary (on/off) keypress actions, our action space also includes mouse movements. As with human gameplay, when in-game GUIs are not open, mouse X and Y actions change the agent's yaw and pitch, respectively. When a GUI is open, camera actions move the mouse cursor. Mouse movements are relative (i.e. they move the mouse or camera relative to the current position), and thus their effect depends on the current position).

Inventory interaction in Minecraft requires fine-grained mouse movements to achieve tasks such as crafting and smelting, while mining and navigating the world can be achieved with coarser mouse action. To be able to achieve both with the same action space, we implemented mouse movement as a set of discrete actions with foveated binning along each axis (Fig. 13), which in preliminary experiments we found to improve crafting performance.

D Inverse Dynamics Model Training Details

D.1 IDM Architecture

The IDM model has approximately 0.5 billion trainable weights. The input to the model is consecutive image frames (128 frames of video), each of which has dimensions 128 × 128 × 3. The IDM is tasked with predicting the action at each frame. All image pixel values are normalized by 255.0 such that they lie within the range $[0, 1]$. The first layer of the IDM is a 3D convolution with 128 learnable filters with a temporal kernel width of 5 and spatial kernel width of 1. The convolution is non-causal, meaning that embeddings at time index t are functions of the frames at times $t-2, t-1, t, t+1$, and $t+2$. We found this layer to be extremely i

Figure 13: Relative camera angle or mouse movement in pixels vs. action bin. The same binning is used for both X and Y coordinates. The binning is foveated, meaning that binning is more fine-grained for smaller movements and more coarse-grained for larger movements. There are 11 bins for each axis (X and Y). The center of each bin (indicated with green circles) is used when un-discretizing movements (that is, when converting from an action expressed as a bin to a camera angle or mouse movement).

Figure 14: Effect of 3-D Convolution in the IDM Architecture.

as it incorporates neighboring temporal information immediately, and we show results comparing IDM performance with and without this layer in Figure 14. This comparison was made on the default (1962-hour) IDM dataset.

This initial temporal convolutional layer is followed by a ResNet[62] image processing network. In this part of the model, no extra temporal information is shared between neighboring frames; however, since each frame was first processed with the temporal convolution, some temporal information is present at this stage. The ResNet image processing network is comprised of three subsequent stacks with widths $W = \{64, 128, 128\}$. Each stack is comprised of, in order, (1) an initial 3x3 convolutional layer with 1-pixel zero padding at the embedding boundary (such that the outgoing embedding dimensions are the same as the incoming embedding dimension) with W output channels, (2) a 3x3 max pooling with stride 2 and padding 1 such that the embedding width and height are halved, and (3) two classic ResNet blocks as defined in He et al.[62] with each layer also having W output channels.

The output of the ResNet stack is flattened into a 1-dimensional vector of size $2^{17} = 131072$ vector for each frame in the video) such that at this stage there are 128 vectors of size 131072 vector is independently processed with two frame-wise dense layers with 256 output activat then 4096 output activations, respectively. The result is then fed through 4 subsequent r (umasked) residual transformer[69] blocks. Each block first has an unmasked attention layer may attend to future frames, with 32 attention heads of dimension 128 each and a surrou connection that skips this layer. The embedding is then passed through a frame-wi with output dimension 16384 and another with output dimension returning to 4096; connection skips past this pair of frame-wise dense layers (not skipping past eac but skipping the pair). All dense layers have their weights tied through time, s video is processed with the same weights.

Finally, independent dense layer heads for each action are pulled from the class on/off categorical parameterized with a softmax for each available

categorical for both the discretized horizontal and vertical mo
details on the action space).

Each dense layer or convolutional layer in the network is preceded by a lay
a ReLU non-linearity. Weights are initialized with Fan-In initialization[71] and b
zero.

D.2 IDM Training

The total loss for the network is the sum of each independent action prediction loss (one for each key and one for both mouse directions). Each independent loss is the negative log-likelihood of the correct action. We use the ADAM[72] optimizer with a linear learning rate decay. We use an initial learning rate of 0.003, a batch size of 128 (where each item in the batch is a video sequence of 128 frames), and a weight decay of 0.01. Hyperparameters were tuned in preliminary experiments. The IDM is trained on our contractor collected dataset for 20 epochs. This took 4 days on 32 A100 GPUs.

We add data augmentation to each video segment; augmentations are randomly sampled once per segment such they are temporally consistent. Using the Pytorch[73] transforms library, we adjust the hue by a random factor between -0.2 and 0.2, saturation between 0.8 and 1.2, brightness between 0.8 and 1.2, and contrast between 0.8 and 1.2. We also randomly rotate the image between -2 and 2 degrees, scale it by a random factor between 0.98 and 1.02, shear it between -2 and 2 degrees, and translate it between -2 and 2 pixels in both the x and y dimensions.

Due the large computational cost of running all of the experiments in this paper, training results are from one run of training (for IDM, BC, and RL training): this non-ideal situation is mitigated because deep learning training tends to be low variance[74,75] and because we often have data points from sweeps (e.g. on dataset size) that suggest overall trends.

D.3 Generating Pseudo Labels with the IDM

Section 4.1 shows that inverse dynamics modeling is a much easier task than behavioral cloning because IDMs can be non-causal. The IDM is trained to simultaneously predict all 128 actions for each video sequence, so the IDM will effectively be causal for frames at the end of the video clip because future frames are not included in the sequence. For this reason, we apply the IDM over a video using a sliding window with stride 64 frames and only use the pseudo-label prediction for frames 32 to 96 (the center 64 frames). By doing this, the IDM prediction at the boundary of the video clip is never used except for the first and last frames of a full video.

E Foundation Model Behavioral Cloning

E.1 Foundation Model Architecture

The behavioral cloning model architecture is the same as the IDM architecture described in Appendix D.1 except that we modify the architecture so that it is causal (i.e. cannot see the future when making predictions). This means the BC architecture does not have the initial non-causal convolution th
IDM has (this layer is omitted completely). Furthermore, the residual transformer layers are n
causally masked (as is standard in language modeling) and we do Transformer-XL-style[76] trai
where frames can attend to keys and values from past batches within the same video. We also
Transformer-XL-style relative attention position embedding.

E.2 Null Action Filtering

The most common action humans take is the null action (no keypresses or mouse move
accounts for 35% of all actions they take. Among other reasons, a player may take th
wait for something in the game to finish, to pause between actions, or to take a bre
of water. Early on in the project we found that the BC model would take a much
35% of null actions, often upwards of 95%. In order to prevent this behavior we fil
null actions from the dataset. We compare a few different treatments: we fil
been 1, 3, or 21 frames of consecutive null actions, and include a treatmen
any null filtering. Null action filtering generally helps, increasing all craft

Figure 15: **(Left)** Effect of Null Action Filtering during training. We compare environment metrics and number of sampled null action during rollouts (rightmost group of columns) for the following treatments: no null action filtering (blue), filtering all null actions (green), filtering only groups of 3 or more null actions (red), and filtering only groups of 21 or more null actions (purple). **(Right)** Hierarchical versus Factored Action Spaces.

Filtering only groups of 3 performed slightly better than filtering all null action or groups of 21. Initial experiments indicated that filtering all null actions was better; however, after further model tuning and after we had already trained our largest models, we found that filtering only groups of 3 or more null actions performed best. Due to compute constraints we were not able to redo all experiments with this setting, but doing so would be a reasonable choice for any future work.

E.3 Joint Hierarchical Action Space

We originally worked with a factored action space, where each keypress could be independently on or off, and this choice was independent of whether the mouse was being moved. This could cause issues for modeling the human behavior distribution exactly. Say for a given state, humans either with 50% probability (a) move forward and attack or with 50% probability (b) move left and drop their item. The best a factored distribution can do is to assign 50% probability to each of the 4 constituent actions because it chooses to press each button simultaneously and independently. See Appendix C.2 for details on the entire action space.

For this reason, we implemented a joint distribution over actions; however, the full joint distribution over 20 binary buttons and two mouse movement dimensions discretized into 11 bins each would result in in $2^{20} \times 11^2 \approx 1.2 \times 10^8$ possible combinations. This is far too large for many reasons, e.g. the final layer from the transformer stack with a dimension of 4096 would need to be mapped to each combination resulting in $4096 \times 1.2 \times 10^8 \approx 5.2 \times 10^{11}$ parameters for this final layer alone. In order to reduce this we noted that many buttons in Minecraft have no effect when simultaneously pressed; for example, if a player tries to move forward and backward at the same time, they remain in place. Below we list the the sets of mutually exclusive actions. Furthermore, the inventory button is exclusive with all other buttons and mouse movement.

Mutually Exclusive Actions
`forward`, `back`
`left`, `right`
`sprint`, `sneak`
`hotbar.[1-9]`

Even reducing the joint action space to reflect these mutually exclusive combinations sti
huge action space when combined with the discretized mouse movements, i.e. $3^3 \times 10 \times$
5.2×10^5. This calculation results from 3^3 for the 3 sets of 2 mutually exclusive ke
taking neither in the set is an option, $\times 10$ for the 9 hotbar keys or no hotbar keyp
remaining binary 4 keys: `use`, `drop`, `attack`, and `jump`, $\times 11^2$ for mouse move
$+1$ for the `inventory` button which is mutually exclusive with all other actions
quite large so we chose to implement a hierarchical binary action for camera
this action is on, then there is a secondary discrete action head with 121 class
of mouse movements because each discretized mouse direction has 11 bin

to move the mouse. If the hierarchical action is off, then t
the secondary mouse movement action is masked during training
need not be sampled during evaluations. While this no longer models
quite a bit better than the factored action space since dependencies betwe
whether or not to move the mouse (although not which mouse movement) are
resulting action space has dimension $3^3 \times 10 \times 2^4 \times 2 + 1 = 8461$ (the 11^2 dimensio
for camera movement has been replaced by a multiplier of 2 here, corresponding to a bi
for whether or not to move the mouse) with an additional 121-dimension head for the joint ca
movements. In the future it would be interesting to implement sequential conditional action spaces to
more completely model the joint distribution.

In Figure 15 (right) we compare environment rollout performance between BC models with the hierarchical joint action space and with the factored action space. Environment statistics are fairly comparable; however, we see that the factored action space model samples far more null actions. This is an important example of the factored action space failing to correctly model the distribution in the dataset because, due to null action filtering, there are 0 null actions in the dataset these models train on. Despite this, the factored model samples many null actions because the prediction for each key is not conditioned on other keypresses.

E.4 Foundation Model Training

The foundation model training is similar to the IDM training, with the exception of labels being IDM-generated pseudo labels. The hyperparameters used for foundation model training are listed in Table 4.

Hyperparameter	Value
Learning rate	0.002147
Weight decay	0.0625
Epochs	30
Batch size	880

Table 4: Hyperparameters for foundation model training

F Behavioral Cloning Fine-Tuning

Behavior cloning fine-tuning is similar to the foundation model training, except we either use a focused subset of all the videos (`early_game` dataset, described in A.3) with pseudo labels, or contractor data (`contractor_house` dataset, described in B.4) with ground-truth labels. The hyperparameters used for behavior cloning fine-tuning are listed in Table 5. We used 16 A100 GPUs for about 6 hours when fine-tuning on `contractor_house` dataset, and 16 A100 GPUs for about 2 days when fine-tuning on `early_game` dataset.

Hyperparameter	Value
Learning rate	0.000181
Weight decay	0.039428
Epochs	2
Batch size	16

Table 5: Hyperparameters for behavior cloning fine-tuning

G Reinforcement Learning Fine-Tuning

G.1 Reinforcement Learning Fine-Tuning Training Details

RL experiments were performed with the phasic policy gradient (PPG) algorith
based on the proximal policy optimization (PPO) algorithm[77] that increas
performing additional passes over the collected data to optimize the val

Hyperparameter	
Learning rate:	3×10^{-4}
Weight decay:	0
Batch size:	40
Batches per iteration:	48
Context length:	128
Discount factor (γ):	0.999
GAE λ:	0.95
PPO clip:	0.2
Max Grad norm:	5
Max Staleness:	2
PPG sleep cycles:	2
PPG sleep value-function coefficient:	0.5
PPG sleep auxiliary value-function coefficient:	0.5
PPG sleep KL coefficient:	1.0
PPG sleep max Sample Reuse:	6
KL divergence coefficient ρ:	0.2
Coefficient ρ decay:	0.9995

Table 6: Hyperparameters for RL experiments. These are the hyperparameters for all treatments with two exceptions. First, when fine-tuning from the early-game model without a KL divergence loss, in addition to the KL divergence loss being set to 0, the learning rate was set to 3×10^{-6} (the best setting out of a sweep over 5 different learning rates), as we found that performance was substantially lower with the standard learning rate of 2×10^{-5} and the agent did not even learn to collect logs. We suspect that the reason that the learning rate needed to be lowered when fine-tuning without a KL loss is that the KL loss prevents making optimization steps that change the policy too much in a single step, especially in early iterations when the value function has not been optimized yet, and the KL loss thus makes it possible to optimize with a higher learning rate. Second, when running RL from a randomly initialized policy there is no KL divergence loss or KL divergence decay, but instead we use an entropy bonus of 0.01, which reportedly worked well in previous work.[30]

protecting the learned skills of the pretrained policy and preventing catastrophic forgetting. As such, we start with a relatively high coefficient ρ and decay it by a fixed factor after each iteration (Table 6). This method protects policy skills in early iterations while guaranteeing that the policy can eventually maximize the reward function, regardless of how different its behavior has to be to do so relative to the pretrained policy.

For the reward function we estimated the rough quantities of each item that a human player might gather when trying to craft a diamond pickaxe, and we reward the model for gathering up to that quantity for each item. We started these estimates by iterating over the technology tree backward from a diamond pickaxe and adding the requirements for each item to the reward function (e.g. first we added a diamond pickaxe to the reward function, then we added the 3 diamonds and 2 sticks required for crafting a diamond pickaxe, then we added the 1 iron pickaxe required for mining diamonds and so on). Then we added coal and torches to the reward function, with coal being useful as fuel when smelting iron and for crafting torches while the torches themselves improve visibility and prevent enemies from spawning. Finally, we reward the model for bringing additional logs (5 logs required to craft all items in the reward function, but we reward up to 8 logs), which can be fuel or crafted into a crafting table or sticks if the agent runs out. In practice the agent rarely uses the additional logs, places the torches, or uses coal as fuel when smelting, but the reward was based on human expectations on what would be useful to execute this task, rather than around how an RL model behaves after training. Finally, to encourage the agent to craft diamonds and crafting diamond pickaxes after it has crafted its first diamond pickaxe, a limit on the number of diamonds or diamond pickaxes that would be rewarded.

The rewards for the different items are separated into 4 tiers, roughly depending on how many would usually get the relevant item. The first tier consists of all wooden and stone items requiring coal with a base reward of 1, the second tier consists of all items requiring iron with a base reward of 4, and the final tier consists of all items requiring iron with a base reward of 4, and the final tier consists of all items later in the sequence of items towards a diamond pickaxe with a base reward of 8. Thus items later in the sequence of items towards a dia

Item	Quantity rewarded
Log	8
Planks	20
Stick	16
Crafting table	1
Wooden pickaxe	1
Cobblestone	11
Stone pickaxe	1
Furnace	1
Coal	5
Torch	16
Iron ore	3
Iron ingot	3
Iron pickaxe	1
Diamond	inf
Diamond pickaxe	inf

Table 7: Reward per item and total quantity rewarded.

give a higher reward. To make sure that the agent does not over-value items that are supposed to be gathered in bulk (e.g. the agent is rewarded for up to 20 planks but only up to 1 crafting table, which can cause the agent to focus on planks at the expense of creating a crafting table), we divide the base reward of each item by the total quantity that the agent gets rewarded for (for the purpose of determining the reward, the total quantity for diamonds is 3 and the total quantity for diamond pickaxes is 1, even though we did not put a limit on the number of these items being rewarded). For example, the agent is rewarded for 3 iron ore, which has a base reward of 4 for being in the iron tier and up to 3 blocks of iron ore are rewarded, thus the reward per block of iron ore is $4/3$. The quantity and reward for each item are listed in Table 7.

While every item in the sequence towards a diamond pickaxe is rewarded, the reward function is still sparse and, in some cases, even deceptive. The sparsity comes from the fact that it can take thousands of actions to find the next reward, even after the agent has acquired all the necessary prerequisites (e.g. human players often take more than 10,000 actions to find a diamond after crafting an iron pickaxe). The reward function can be deceptive when the most efficient method for getting one item can make it far more difficult to get the next item. For example, a good strategy for the agent to craft a stone pickaxe quickly is to mine (i.e. spend a few seconds to pick up) its crafting table after crafting a wooden pickaxe, such that the agent has immediate access to a crafting table as soon as it has collected enough cobblestone. However, the fastest way to get a reward for gathering cobblestone is to mine down immediately after crafting a wooden pickaxe, while leaving the crafting table behind. Thus following the optimal strategy for gathering cobblestone makes it more difficult to learn to craft a stone pickaxe.

Experiments ran for approximately 6 days (144 hours) on 80 GPUs (for policy optimization) and 56,719 CPUs (mostly for collecting rollouts from Minecraft). In this time the algorithm performed roughly 4,000 optimization iterations and collected roughly 1.4 million Minecraft episodes consisting of 12,000 frames each, for a total of 16.8 billion frames.

G.2 Reinforcement Learning Fine-Tuning Additional Data

Additional figures that are helpful for understanding the main results of the RL fine-tuning experiments are presented in this section. First, we show the items-over-training figure when RL fine-tuning the early-game model without a KL loss (Fig. 16). When training without a KL loss, the agent only learns to obtain the four items that the early-game model is capable of getting zero reward for: logs, planks, sticks, and crafting tables. Second, we present preliminary experiments where we directly compare RL fine-tuning from the house-building model and RL fine-tuning from the early-game model (Fig. 17). These experiments differ from the main experiments in a few ways. First, the treatments shown here, the KL loss coefficient was set to 0.4, the learning rate was $1/3$ of the main experiments, and the reward for each item was $1/quantity$ for all items (i.e. items closer to the house did not have an increased reward). While RL fine-tuning from the house

worked better than RL fine-tuning from the early-game model, worked better after 800,000 episodes and showed signs of smelt early-game model was chosen for the main experiments.

Figure 16: Items obtained when RL fine-tuning from the early-game model without a KL loss. The model learns to obtain all items that the early-game model can craft zero-shot, which are logs, planks, sticks, and a crafting table. In contrast to the treatment with a KL-penalty, it does not learn any items beyond these initial four, likely because skills that are not performed zero-shot, and for which the model thus does not initially see any reward, are catastrophically forgotten while the first four items are learned.

Figure 17: Preliminary experiments when RL fine-tuning from the early-game model compared to RL fine-tuning from the house-building model. **(Left)** While reward initially increases faster when fine-tuning from the house-building model, fine-tuning form the early-game model eventually obtains a slightly higher reward. **(Right)** RL fine-tuning from the early-game model has a higher likelihood of smelting an iron-ingot, which is why the early-game model was chosen for future RL fine-tuning experiments.

H Foundation Model Scaling

In early experiments we found that increasing model size led to models staying in the efficient learning regime longer into training.[63] Here we compare the 0.5B model described in Section 4.2 to both a 248M and 71M parameter model. Both of these models are trained for 15 epochs as compared to the 30 epochs the 0.5B model trained for. These models have the same architecture as the 0.5B model but each layer in the 248M parameter model has 1/2 the width and each layer in the 71M parameter model 1/3 the width. The 71M model was trained with an initial learning rate of 0.0015, batch size of 480, and weight decay of 0.044506. The 248M model had an initial learning rate of 0.001831, batch size of 640, and weight decay of 0.051376.

In Figure 18 we show validation loss on `web_clean` with IDM pseudo-labels, loss on the c dataset used to train the IDM with ground truth labels collected during contractor play, and environment performance for the 71M, 248M, and 0.5B models. While larger models validation loss on `web_clean`, these results do not tell the clear story that the 0.5B m than its smaller counterparts. The 71M model has the lowest contractor dataset loss w highest `web_clean` loss, and it also has the best zero-shot environment performance that the 71M model even had non-zero wooden tool crafting (Fig. 18 bottom left), also appears to be better at crafting than the 0.5B, and also has lower contractor

While the zero-shot results suggest smaller models are better, fine-tuning tell fine-tuning to `contractor_house`, model size rank ordering reverses an performs best both in validation loss (Fig. 19 left) and in environment pe

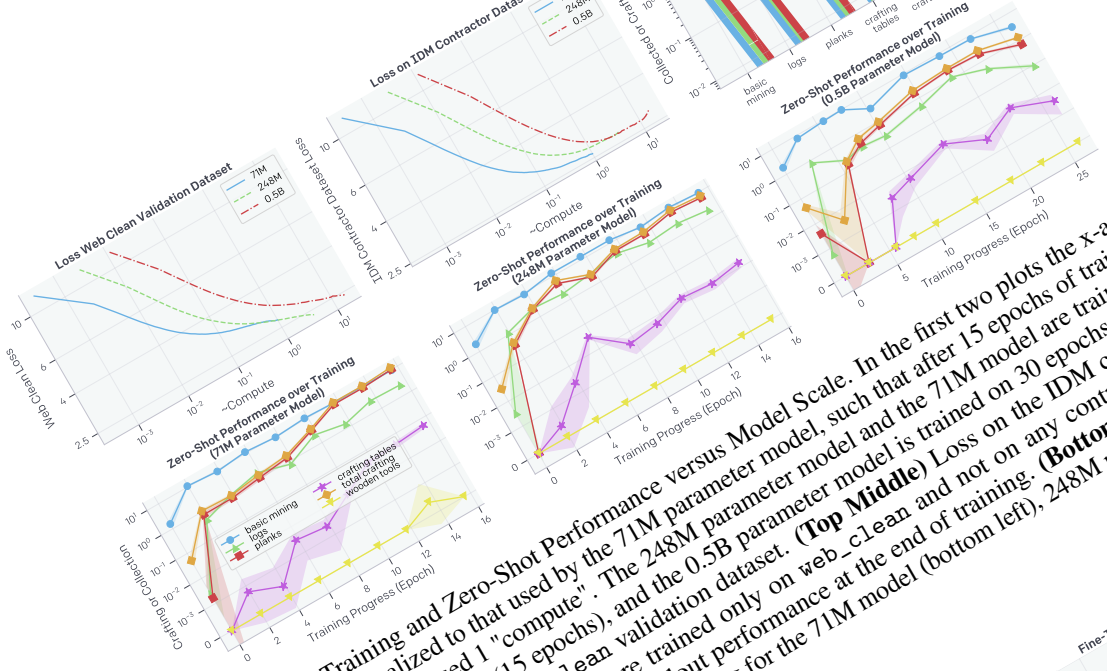


Figure 18: Training and Zero-Shot Performance versus Model Scale. In the first two plots the x-axis is compute normalized to that used by the 71M parameter model, such that after 15 epochs of training the 71M model has used 1 "compute". The 248M parameter model and the 71M model are trained on the same amount of data (15 epochs), and the 0.5B parameter model is trained on 30 epochs of data. **(Top Left)** Loss on the `web_clean` validation dataset. **(Top Middle)** Loss on the IDM contractor dataset; note that these models were trained only on `web_clean` and not on any contractor data. **(Top Right)** Zero-shot environment rollout performance at the end of training. **(Bottom)** Zero-shot environment rollout performance over training for the 71M model (bottom left), 248M model (bottom middle), and 0.5B model (bottom right).

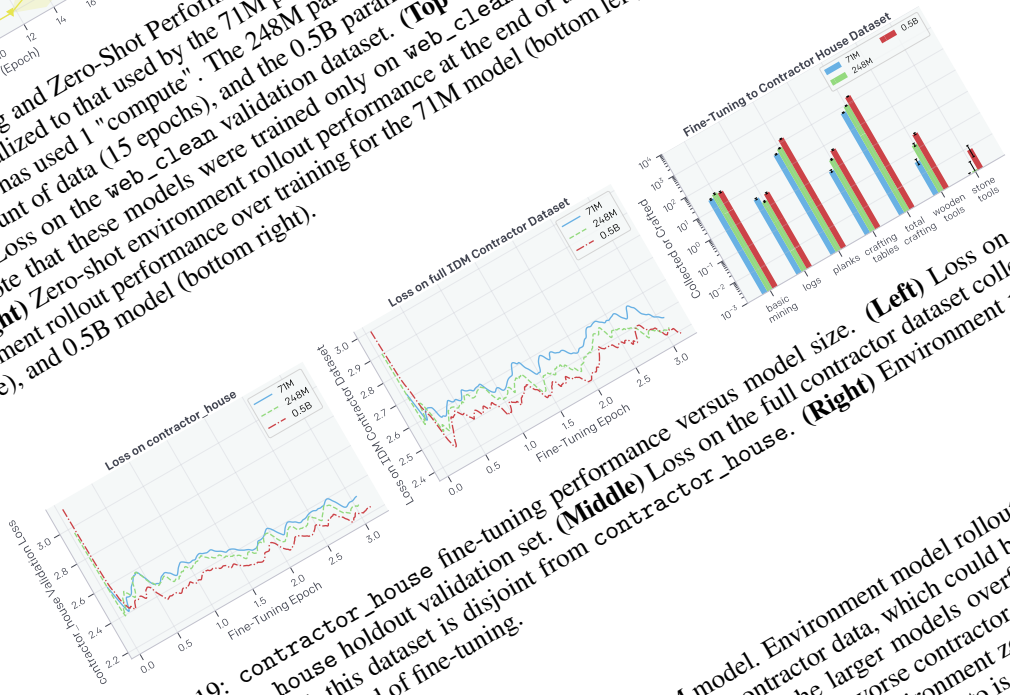


Figure 19: `contractor_house` fine-tuning performance versus model size. **(Left)** Loss on the `contractor_house` holdout validation set. **(Middle)** Loss on the full contractor dataset collected to train the IDM; this dataset is disjoint from `contractor_house`. **(Right)** Environment rollout performance at the end of fine-tuning.

followed by the 248M model and then the 71M model. Environment model rollouts are perfor
using the same game engine that we use to collect contractor data, which could be visually di
from videos taken from the web. It is plausible that the larger models overfocus on the
peculiarities in web data during pretraining since they have worse contractor data loss (F
middle), and this causes them to perform more poorly in the environment zero-shot. Ho
hypothesize that because the `contractor_house` dataset we fine-tune to is collected fro
engine, the larger models that are a better overall Minecraft prior (as indicated by lowe
validation loss in Fig.18 top left) can quickly shift their low level features to perform
coming from our game engine, resulting in better environment rollout performance
is further supported by Fig. 19 (middle) showing loss on the contractor dataset
training, which has no overlap with `contractor_house`. After just a few ste
`contractor_house`, all models quickly improve in loss on the full IDM co
larger models now performing best. While not conclusive, we believe this
some intuition for future studies of model scaling for sequential decision m

I Text Conditioning

Goal-conditioned policies[81,82] make it possible for a single agent to perfo goals in a single environment, which is particularly relevant in open-ended envi Minecraft. In recent work, goal specification has increasingly taken the form of do languages[83], or even natural language[84,85]. The benefits of language-conditioned agents tremendous, especially *natural*-language-conditioned agents, as their goal space contains a w variety of potentially very complex tasks. Text conditional models have shown an amazing ability to perform tasks zero-shot (or learn them few-shot) including generalizing in impressive ways via the compositional and combinatorial possibilities allowed by natural language (e.g. GPT[1] and DALL·E 2[86]). We hypothesize that we should expect similar capabilities to emerge with natural-language-conditioned virtual agents, if they are similarly trained on enormous amounts of data (that goes from a natural language description to a sequence of actions that completes the specified goal). In this section we take preliminary steps toward that future. Our preliminary experiments provide evidence that it is possible to pretrain a natural-language-conditioned model for Minecraft using the general approach presented in this paper (VPT) plus conditioning on the speech that often accompanies videos.

In online videos, the human actor sometimes indicates their intent in their verbal commentary (e.g. “Let’s go chop some trees to make a wooden axe” or “now let’s learn how to crop photos in Photoshop”). Conditioning on this closed caption data could produce a *steerable* pre-trained model: i.e., it may later be possible to condition the model with text such as “I am going to craft a wooden pickaxe” or “I am going to build a house,” and have the agent perform those tasks specifically rather than simply follow typical human behavior (as was investigated in the rest of this paper). An alternate way to produce a steerable agent is via RL fine-tuning, which we could have done in Section 4.4 by adding a bit indicating the task to be completed, as has been done in prior work[30]. However, conditioning on natural language offers many benefits over that approach. First, it is flexible and powerful, being able to express any task. Second, one does not need to preconceive of the task to be completed ahead of time. This would allow for general, capable, zero-shot agents like GPT, but extending those capabilities to embodied tasks such as completing tasks on computers or in simulated 3D worlds. Third, text conditioning can be used even when tasks are difficult to specify via reward functions (e.g. “Let’s build a house” or–if the agent is capable of doing it–more complex things like “I will now build a castle surrounded by a moat”). In the limit, VPT+text could conceivably produce powerful, capable, natural-language-conditional agents with the powers of GPT to meta-learn, follow instructions, and complete tasks zero or few shot, but in the form of agents that can act in virtual worlds, complete tasks on computers, and in other similar embodied sequential decision domains. We do not reach those lofty goals in this work, but we began a first step towards exploring in that direction.

Many Minecraft videos feature audio commentary from the player. This commentary is sometimes present in the form of closed captions for the videos, or could be extracted post-hoc using automated speech recognition (ASR).[87] Our dataset features about 17k hours of content with associated closed captions.

We fine-tuned the 220 million parameter VPT foundation model used in the RL-fine-tuning e periments (chosen vs. 0.5B for the same reason: to reduce compute costs) with an additi text-conditioning input on the subset of our data for which closed captions are available. To obta conditioning input, we first split videos into 30 second chunks. The same text is associated wit frame in a given chunk, and is made up of all the closed captions occurring within that chunk as the line of text preceding and following the chunk (if any). Because the vast majorit 95%) of our closed caption data lacks capitalization and punctuation, it is punctuated usin library[88]. We then obtain a text embedding vector of length 4,096 from the OpenAI emb which is processed by a randomly initialized multi-layer perceptron (MLP) with two h size 2,048. The resulting activations are added for each frame to the pretrained m before the transformer layers (`pretransformerActivations += mlp(textE` model is fine-tuned for four epochs.

Our model shows evidence of steerability. When conditioned on sentences t explore (such as “I’m going to explore” and “I’m going to find water”) the a farther from its spawn point (Figure 20a). Additionally, we can steer the age

Figure 20: Evidence for conditioning. In each plot, the variants expected to stand out are shown in bold. The strings corresponding to each variant are shown in Table 8. Statistics are measured over 5 minute episodes. **(a)** Distance traveled by the agent . Both "explore" and "water" text strings should encourage a steerable agent to move more than when doing other tasks, which is what occurs. Grass (which is needed to get seeds) is not present in all biomes, which is likely why the "seed" condition produces more travel (as the agent sometimes needs to move to a biome with grass). The travel distance is the Euclidean distance from the spawn point to the farthest point the agent reached during the episode on the horizontal (x-z) plane. **(b)** Collection of wheat seeds. The "seed" variant collects substantially more than other variants, as expected of a steerable agent. **(c)** Collection of oak (the most common type of wood) logs. The "wood" variant collects significantly more oak logs, as is to be expected of a steerable agent (we speculate that the "water" variant collects less because there are no trees in water). **(d)** Collection of dirt. The "dirt" and "dig" variants collect a large amount, and are the variants that are (indirectly in the case of "dig") conditioned to collect dirt. It is easy to mistakenly aim at the ground rather than at grass or trees when collecting seeds or wood, which likely explains the slightly higher amount of dirt collected by these variants. In all cases, the error bars are 95% confidence intervals of the mean, over 1,000 episodes per conditioning variant. Treatments for which the bars in each bar plot do not overlap are statistically significantly different at a $p < 0.05$ level.

Variant name	String
dig	I'm going to dig as far as possible
dirt	I'm going to collect dirt
explore	I'm going to explore
house	I'm going to make a house
seed	I'm going to collect seeds
water	I'm going to find water
wood	I'm going to chop wood

Table 8: Strings corresponding to each conditioning variant

early game items such as seeds, wood, and dirt by conditioning

seeds/chop wood/collect dirt" (Figure 20b,c,d).

While our results show some level of steerability, more work is required t

we were not able to successfully steer agents to gather flowers or to hunt, both

in the early game, but less common (and, in the case of hunting animals, much m

gathering dirt, wood, or seeds. Likewise, an experiment in which the agent is pres

crafting window and various resources, and conditioned to craft a given item (e.g. "I m g

craft a wooden axe") failed to show that the conditioning had a significant effect on which items

crafted. Instead, it seemed the agent was more influenced by the prior, unconditional probability

of what human players would craft next given the resources available, which is not too surprising

since in Minecraft, especially in the early game, there is a relatively consistent path to gathering

resources in a specific order go produce more powerful tools (Fig. 6). For example, if the agent had

the resources to make a stone pickaxe and we asked it instead to make a (weaker) wooden pickaxe, it

often would make the stone pickaxe anyway. Finally, looking at videos of agent behaviors failed to

convince us that the "house" conditioning causes the agents to take more steps towards building a

house than other variants.

Thus, our results show that it is possible to train a somewhat steerable natural-language-conditioned

agent. However, its steerability is still too weak to be practically useful, and it is far from what we

believe could be accomplished with more research, data, and training compute. Another exciting

research direction is to have the model predict future text as well as just the next action.