MultiSTEVE-1s: A Model Zoo and Interpretability Suite for Instruction-Following Vision Agents

Karolis Jucys
University of Bath
k@rol.is

George Adamopoulos
McGill University
george.adamopoulos@mail.mcgill.ca

Özgür Şimşek
University of Bath
o.simsek@bath.ac.uk

Abstract

A striking case of goal misgeneralisation was previously observed in OpenAI's Minecraft agent VPT: it killed villagers standing under leaves, mistaking them for tree trunks. Although this agent was released publicly, enabling white-box interpretability research, few open-weight model organisms of misalignment exist outside the LLM space. We release MultiSTEVE-1s: a model zoo of 140 fine-tuned versions of VPT, over 1,000 training checkpoints, and an interpretability suite for analysing them. We use the STEVE-1 training procedure to add instruction-following capabilities to VPT with fixed hyperparameters and controlled variations in training randomness. We demonstrate the utility of MultiSTEVE-1s by showcasing the research it enables. First, some training runs differ only by a least-significant bit flip🦋 in a single initialised weight. Others differ in the full randomness for weight initialisation and data. Yet, the single bit-flip setting produces agents that act nearly as differently🌪️ from each other as the full randomness ones. Second, we use our interpretability suite to show that several known VPT attention heads retain their roles after STEVE-1 fine-tuning, while attention strength to the same behaviourally meaningful frame can vary substantially across agents and checkpoints. Finally, although the agents are similarly capable on in-distribution tasks, their out-of-distribution behaviour of villager killing can differ substantially: in one setting, one agent kills villagers less than 5% of the time, while another kills them nearly 50% of the time. Our results show the value of studying multiple similarly trained agents rather than acting like a behavioural biology lab with only one rat.[1]

1 Introduction

A recent position paper argues that two distinct models can perform equally well on their training data, yet generalise completely differently out of distribution [1]. This has direct implications for alignment: a model that appears well aligned during evaluation may or may not act misaligned out of distribution, depending on the training random seed. Open-weight models of large-scale agents for complex interactive virtual worlds are scarce, limiting empirical study of out-of-distribution behaviour. Well-known examples such as AlphaStar [2], OpenAI Five [3], Gato [4], Ada [5], and SIMA [6] are all non-public. VPT [7] and STEVE-1 [8] are among the few public ones.

We release MultiSTEVE-1s: 140 STEVE-1 agents trained with fixed hyperparameters and controlled randomness across five groups: weight initialisation seed only, data seed only, both seeds, a single least-significant-bit flip in one MineCLIP embedding-layer weight before training, and a larger both-seeds group for outlier detection.

We provide empirical support for several claims from the position paper, building on prior findings that VPT exhibits goal misgeneralisation and attacks a brown villager under the foliage of a tree

[1] Models: HF models; data: HF dataset; attention dashboard online demo: multisteve-1s.netlify.app.

Mechanistic Interpretability Workshop at the 43rd International Conference on Machine Learning, Seoul, South Korea, 2026.
Copyright 2026 by the author(s).

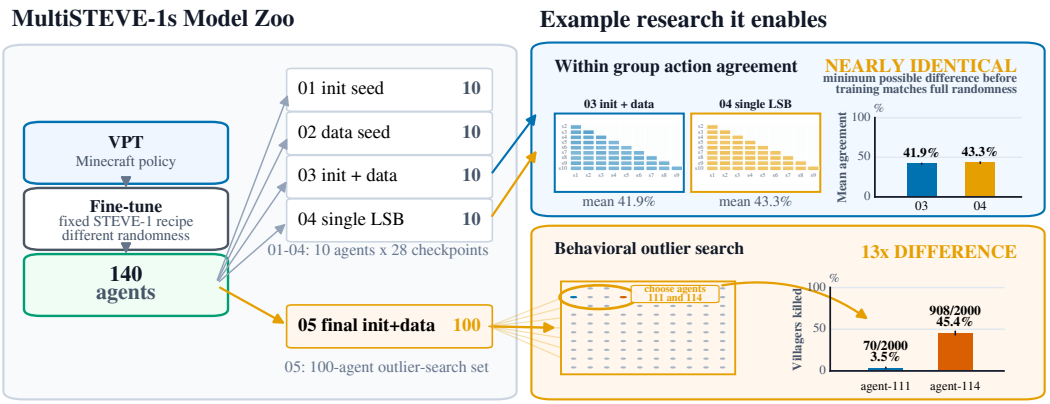


Figure 1: MultiSTEVE-1s consists of 140 agents trained with fixed hyperparameters and different randomness. It enables research that would be impossible with single-agent model releases. Example finding: a single least-significant-bit (LSB) flip in one weight before training produces as much diversity between trained agents as full randomisation. This diversity then amplifies differences in out-of-distribution goal-misgeneralisation behaviour: in one setting, one agent kills villagers 13 times more often than another otherwise similarly trained agent.

[9–11]. We study whether this example of misgeneralisation occurs at different rates across the nondeterminism sources above.

We offer the following contributions:

1. A model zoo of 140 STEVE-1 agents with more than 1,000 checkpoints, extensive logs, training code, and datasets.
2. An interpretability suite with an attention pattern dashboard, rollout infrastructure, a 3D world and trajectory viewer, and other tools for studying these agents.
3. Examples of enabled research, including evidence that training seed alone can substantially change villager-killing frequency, and that flipping a single least-significant bit in one weight before training can produce as much divergence in final model actions as fully randomising weight initialisation and data.

Our checkpoints and training data are available at Hugging Face Models and Hugging Face Datasets; the training code can be found in the supplementary material.

2 Related work

Underspecification and model zoos. A recurring finding in machine learning is that a single training pipeline can produce many models with equivalent in-distribution performance, but with wildly different behaviour under distribution shift [12–14]. This phenomenon is known as underspecification [15]. MultiBERTs [16], Pythia [17], and PolyPythia [18] address it by releasing many models trained with controlled randomness, enabling more robust conclusions about model behaviour.

Training dynamics and nondeterminism. Very small changes in model initialisation, including single-bit-level changes, can compound over training to produce substantially different final weights [12]. Singular learning theory offers a theoretical argument for why this occurs. It posits that models with identical in-distribution loss can occupy different loss basins and therefore generalise differently out of distribution [1, 19, 20]. Loss landscape flatness has been shown to correlate with generalisation ability, offering one lens through which to understand why models can diverge from nearly identical starting points [21]. Security researchers have also studied bit-flip sensitivity, showing that flipping a single bit can cause up to a 90% drop in classification accuracy on a particular task [22].

Goal misgeneralisation. Goal misgeneralisation has been formalised as a robustness failure in which an agent competently pursues an undesired goal that leads to good performance in training situations

but bad performance in novel test situations [10]. The first explicit empirical demonstrations of this phenomenon in deep reinforcement learning appeared in prior work on goal misgeneralisation [11]. Empirical studies have found instances of this in a range of settings [23–25]. We focus on a prior finding that the VPT model misgeneralises to attack a brown villager under the foliage of a tree [9]. Goal misgeneralisation builds on the broader generalisation literature [26–30].

Mechanistic interpretability. Mechanistic interpretability aims to reverse engineer the internal computations of neural networks by analysing activations, attention patterns, weights, and circuits [31–33]. Attention-head analysis has an established methodology for characterising the role of individual heads [34, 35]. Most work in this area focuses on large language models, with limited coverage of vision-based agents [36, 9]. A recurring challenge is the scarcity of open-weight model organisms suitable for controlled study [37]; most existing work on model organisms focuses on LLMs [38]. Our intermediate checkpoints are also intended to support developmental interpretability [39], which studies how a model's representations and circuits form over the course of training.

3 Model zoo

The model zoo is built on VPT [7], Minecraft agent pre-trained on pseudo-labelled internet-scale gameplay data and fine-tuned using reinforcement learning to obtain diamonds. The network architecture consists of a convolutional neural network to embed the pixel observations, and a transformer with 128 frames (6.4 seconds) of memory. The agent uses a keyboard and mouse action space. STEVE-1 adds text and visual instruction following to VPT using MineCLIP [8, 40].

Computational resources. We trained our agents on a single GPU with 48 GB of VRAM, but training is possible on a GPU with 24 GB of VRAM by reducing the batch size and increasing gradient accumulation threefold. Rollouts can be generated using only 3.5 GB of GPU VRAM. In total, the project used roughly 11,500 GPU hours, resulting in 1–2 tonnes of CO_2 emissions; see Appendix I for full details.

3.1 Training

We built on the original STEVE-1 codebase to train our model zoo, and implemented comprehensive randomness controls. Like STEVE-1, we fine-tune the weaker `rl-from-foundation-2x` checkpoint of VPT, which does not obtain diamonds. We also experimented with fine-tuning the strongest VPT checkpoint, `rl-from-early-game-2x`, and observed broadly similar results. Each model is trained on one NVIDIA L40S GPU, rather than the four NVIDIA A40 GPUs used to train the original STEVE-1 agent. See Appendix E for the measures we took to control and account for randomness, and Appendix G for the full list of differences between the original training procedure and ours. All runs use the same hyperparameters. We provide all training logs and data-order logs.

Types of models. The model zoo includes five different groups of models:

- 01_init_seed_only: only the weight initialisation seed varies.
- 02_data_seed_only: only the data seed varies.
- 03_init_and_data: both seeds vary.
- 04_single_lsb_flip: the random seeds are fixed, but we flip a single least-significant bit in a different randomly chosen MineCLIP embedding-layer weight.
- 05_final_init_and_data_100: both seeds vary, with 100 additional models for outlier hunting.

We provide 10 models with 28 checkpoints each for groups 01–04. For group 05, we provide 100 models with only the initial and final checkpoints. If needed, we could rerun the deterministic training procedure and save missing intermediate group 05 checkpoints or new metrics for any agent. For group 04, we flip the 14th bit from the end of the mantissa, which is equivalent to a least-significant-bit flip because TF32 operations discard the last 13 bits; see Appendix F for more details. Group 04 represents the smallest possible numerical difference between two models before training.

Randomness. We fix the sources of randomness in the training procedure via two seeded random number generator (RNG) streams for each training run. The first RNG stream is used to initialise the MineCLIP embedding layer added to VPT during the STEVE-1 training procedure. The second stream controls the data: its order, hindsight goal relabelling, and classifier-free conditioning dropout.

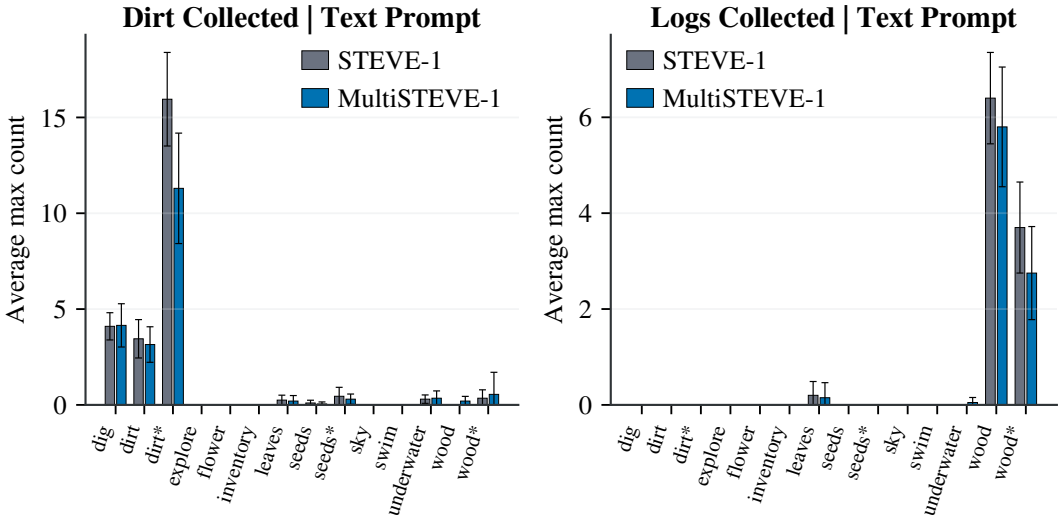


Figure 2: Comparison of the original STEVE-1 checkpoint and our MultiSTEVE-1 model. Our model matches the original: agents primarily perform the task specified by the text prompt. See Appendix K for the exact text prompts.

Full details of the original training procedure are given in the STEVE-1 paper [8]. For our bit-flip model group, we use an additional RNG stream to choose which of the roughly one million weights in the MineCLIP embedding layer will have its least significant bit flipped.

We use multiple checks to verify that we control and account for all randomness in the training procedure, such as confirming that global RNG seeds have no remaining impact, verifying that identical seeds produce bitwise identical metrics and weights, extensively logging metrics and data order, and saving initial checkpoints. See Appendix E for full details.

Data. The original STEVE-1 [8] used two datasets to train its policy: OpenAI Contractor Dataset [7] and VPT-Generated Dataset, with 39 and 15 million frames, respectively. The authors did not share the VPT-Generated Dataset or the MineCLIP embeddings for either dataset. We collected the VPT-Generated Dataset and embedded both datasets using the authors' shared code. This cost roughly 550 GPU hours, or about $170. This was not included in the $60 headline training cost reported by the authors. To help speed up future work, we release the full embedded datasets subject to documented terms.

3.2 Evaluation

We follow the original STEVE-1 programmatic evaluation protocol [8]. This includes running rollouts in the MineRL 1.0 environment [41] and measuring travel distance and obtained item counts. To reduce evaluation noise, we reduce episode length from 3,000 to 1,000 frames, increase the number of episodes from 10 to 20, and fix randomness in the text-prompt embedding and sampling process. Unless otherwise stated, error bars, shaded bands, and $\pm$ values show two-sided 95% confidence intervals: mean metrics use normal/t-based intervals for the mean, binary rates use Wilson score intervals, and intervals are clipped to valid metric ranges where applicable.

Full evaluations. We perform full evaluations using 14 text and 11 visual prompts on the original STEVE-1 checkpoint and on one trained by us with random seeds set to 1 and 1. This reproduces the original STEVE-1 programmatic evaluation results on both model checkpoints [8]. As in the original paper, the agents obtain relevant items much more reliably when given the corresponding prompt; for example, they obtain wooden logs with the text prompt 'chop a tree' (see Figure 2). See Appendix A for the full results. It also shows that the measured performance of our trained model broadly matches that of the original one.

Partial evaluations. Since complete evaluations on the whole model zoo are computationally prohibitive, we reduce the number of prompts from 25 to 4 when evaluating the four groups of ten models each. We use the dirt and wood visual prompts and the engineered dirt and wood text prompts

('chop down the tree, gather wood, pick up wood, chop it down, break tree' instead of 'chop a tree', marked wood*). The results are somewhat noisy, but performance is relatively stable across all 40 models on these tasks; see Appendix B. We also performed the same partial evaluations on 28 checkpoints throughout the training of a single model. The results show a relatively noisy increase in performance metrics over time; see Figure 9 in the appendix.

Minimal evaluations. For the final group with 100 trained models, we choose only the hardest task from the evaluation set: chopping trees with a text prompt. This lets us check whether any models are unable to solve this task. All 100 models can reliably find and chop trees, although there may be differences in the mean number of logs obtained; see Figure 24 in the appendix. When qualitatively inspecting episode rollouts, the models act very similarly on these in-distribution tasks.

Overall, there are no large differences between the 140 final model checkpoints in our model zoo, when evaluated on in-distribution tasks.

4 Interpretability suite

As part of the release, we open-source the tools used for our interpretability work. They were designed for quick customised experiment rollout generation, 2D and 3D trajectory and environment visualisation, and inspection of the agent's attention-head activation patterns, action probabilities, and other metrics. We welcome further contributions from the community.

Attention patterns. Inspired by prior attention-pattern analysis of VPT [9], we build a new interactive attention-pattern analysis dashboard. It shows attention patterns, current and attended-to frames, camera-action probability distributions, and metrics at the corresponding time and attention position. The demo is available at https://multisteve-1s.netlify.app/; see Figure 17 in the appendix.

Rollout generation. We provide extensible code for creating scenarios with different regular trees, villager-trees, object counts, positions, and nearly 100 villager types. We also provide configurations to queue up thousands of episodes. Preset logging options range from lightweight statistics collection to full per-frame logging of observations, action probabilities, and attention activations.

3D trajectory and environment visualisation. In addition to the simple top-down 2D trajectory visualisation, we provide an interactive 3D viewer of both the environment and replays of agent trajectories; see Figure 18 in the appendix.

Environment seed library. To help set up experiments, we provide full-quality first-observation images for over 17,000 environment seeds here. In subsection L.4 , we give example seeds, list the seeds used for the T-maze experiments, and list seeds with visible nearby villages.

Other. We also provide interactive scatter plots for T-maze analysis and faster short-rollout generation using agent teleports instead of full environment resets; see our codebase for full details.

5 Results

This section gives examples of research enabled by our model zoo and interpretability suite. The findings are preliminary and exploratory, demonstrating interesting phenomena and inviting further research rather than giving firm conclusions for why they occur.

5.1 Action probabilities

At each timestep, the agent outputs a probability distribution over actions, from which an action is sampled stochastically. In some cases, the next action is obvious: for example, standing still and attacking when the agent is in front of a tree it has already attacked for 50 consecutive actions. In other cases, the next action is almost arbitrary: when the agent sees no trees and is told 'chop a tree', turning left or right to look for one is equally good. In the latter case, trajectories can differ substantially even when the agent is placed in the same scenario and given the same instructions.

To measure differences within the four groups, we compare deterministic action probabilities on fixed data rather than stochastic model-controlled rollouts. We produce 14 rollouts of 1,000 steps each, one for each text prompt, using the model trained with seeds set to 1 and 1. We then have each model produce action probabilities for each frame sequentially. From the agent's perspective, it appears to

Table 1: Pairwise agreement across nondeterminism sources. Exact agreement means that the two agents' highest-probability actions match exactly: the same button action out of 8,641 possible actions and the same camera action out of 121 possible camera actions. Semantic button agreement maps joint button actions into groups such as forward/back, left/right, attack, and jump, then calculates the similarity between probabilities assigned to each group.

	Pairwise agreement (%)	
Nondeterminism source	Exact	Semantic button
Single Least-Significant-Bit Flip	43.26 ± 0.91	91.30 ± 0.16
Weight Init Seed Only	43.26 ± 0.65	91.29 ± 0.13
Data Seed Only	41.86 ± 0.75	90.91 ± 0.16
Weight Init and Data Seeds	41.87 ± 0.59	90.91 ± 0.14

be taking actions in the real environment. However, its actions are ignored, and it sees the next frame of the fixed rollout. Each agent's 128-frame memory is populated with these frames.

Table 1 shows the mean pairwise agreement between models trained using each nondeterminism source, sorted from weakest to strongest perturbations. Flipping the least significant bit in one of 250 million weights produces about as much difference between final trained models as fully randomising the data and the weight initialisation. This mirrors previous bit-level nondeterminism results [12], except that in our case we are only fine-tuning a model instead of fully retraining it. This adds evidence that neural networks are ~~cursed~~ best viewed as chaotic systems [42].

5.2 Attention patterns

The transformer part of VPT's network uses attention-based memory of 128 frames, equivalent to 6.4 seconds of real-time gameplay at 20 frames per second. There are four layers with 16 attention heads in each layer. We refer to the post-softmax, input-dependent attention matrix as the attention pattern (sometimes called attention weights in the literature) to distinguish it from the model's static learned parameters. Inspecting these patterns can suggest the function of each attention head.

Prior work identified attention head 0.9[2] as important for VPT [43]. Later attention-pattern analysis found that head 0.9 nearly always pays the most attention to the previous frame, that head 0.8 acts as an indicator for the inventory menu being open, and that head 2.2 is related to upward camera movement [9]. We extend these findings to our STEVE-1 agents, which are instruction-tuned versions of VPT, and confirm that attention heads 0.8, 0.9, and 2.2 retain their functions. We also find that attention head 2.2 responds to jumping actions, and that head 1.3 is related to looking or moving to the right. All models in our model zoo retain these broad attention-head functions.

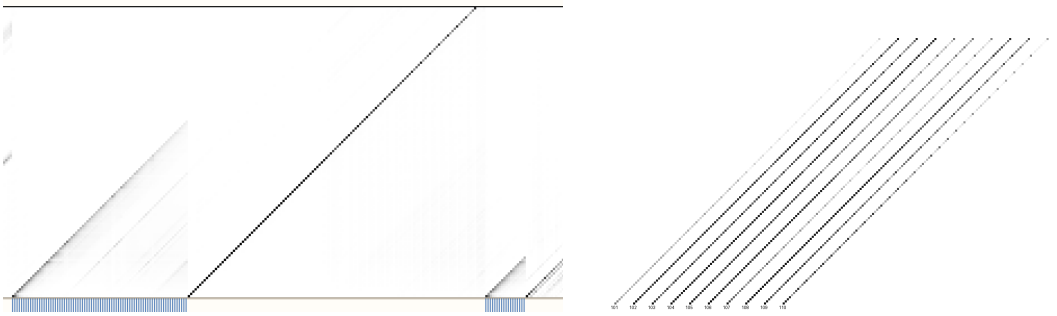
However, we find attention-pattern differences between our models. For example, in one tree-chopping scenario, attention head 1.14 of Agent-1 attends strongly to the last frame in which the inventory menu is open.[3] This appears as a bright diagonal line at frame 802 in the attention-pattern plot; see Figure 3a or the default view in our online demo. We then show exactly the same rollout frames to agents 101-110. Every agent pays attention frame 802, but the average attention weights vary between 0.16 and 0.83; see the dim line of Agent-101 and the bright line of Agent-104 in Figure 3b. When we showed the episode to all 28 training checkpoints of Agent-1, we found that this attention-pattern line gets brighter over training, but surprisingly it is the brightest at training step 20M instead of the final 160M checkpoint; see Figure 15 in the appendix.

5.3 Out-of-distribution behaviour

Prior work found that VPT often kills villagers standing under tree leaves [9]. We use the same setup, which includes four invisible barrier blocks preventing the villager from moving, to reproduce and extend this finding to STEVE-1 agents, which also kill villagers standing under tree leaves when prompted to 'chop a tree'. Because the agents correctly chop regular trees, this is not reward misspecification [44], but goal misgeneralisation [10, 11]: instead of learning to chop trees, the agent

[2]'attention head 0.9' means head 9 in layer 0.

[3]'Agent-1' means the agent trained with weight initialisation seed 1 and dataset seed 1.



(a) Attention pattern + inventory state viewer

(b) Frame 802 attention patterns, 10 models

Figure 3: (a) Attention pattern for head 1.14 in one episode around timestep 802. The bright diagonal line shows the agent assigning nearly all attention to a single frame: the last one with the inventory menu open. The blue bars at the bottom correspond to the inventory state. (b) The same diagonal when the episode is shown to ten different agents: Agent-101 pays much less attention to the frame than Agent-104, 0.16 and 0.83 mean attention strength respectively.

appears to have learned to chop vertical brown objects under green leaves. The distinction becomes apparent only in the out-of-distribution setting, where the tree trunk is artificially replaced by a villager wearing brown clothes. Next, we demonstrate how our agents can exhibit significantly different misgeneralised behaviour in various out-of-distribution settings. First, we explore a T-maze-inspired setting in which agents choose between a regular tree and a villager-tree across several scenarios [45–47]. Second, we find outliers in out-of-distribution behaviour: in one setting, Agent-111 kills villagers 3.5% of the time, while Agent-114 kills them 45.4% of the time.

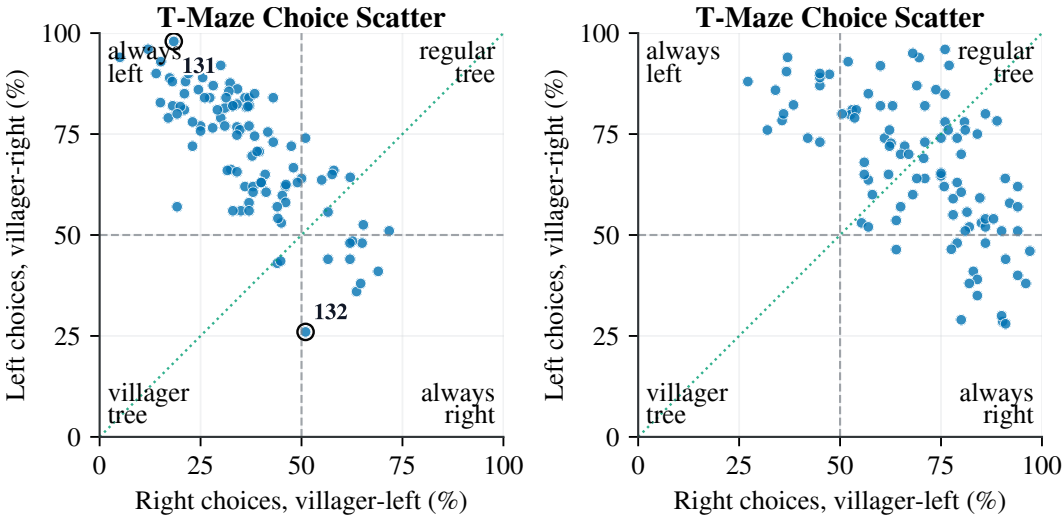
T-maze. The T-maze setting is designed to make us zoom out. We place two equidistant objects in front of the agent: a regular tree and a villager-tree, as shown in Figure 4. We use two environment seeds: seed 49, where a brown hill on the right makes the scene visually asymmetric, and the more symmetrical seed 481. Now start with the smallest possible experiment. We roll out one agent once, and in Figure 6a we see a single blue trajectory going to the bottom left. If that outlier trajectory were the only observation we had of STEVE-1's behaviour, would we conclude that 'STEVE-1 ignores the scientists' task and leaves to do its own thing'? Probably not. We need to observe it in the same situation many times.

So we do that. We run 100 three-second rollouts, long enough for the agent to choose and approach one of the two objects. For Agent-131 in environment seed 49, with the regular tree on the left and the villager-tree on the right, this shows that the agent chooses the regular tree 98% of the time; the blue trajectory was just an unlikely outlier. So, clearly, 'STEVE-1 does not fall for the scientists' trick and acts morally 98% of the time'? Closer, but not quite there yet. We have only studied one object ordering in one background. When we swap the objects, Agent-131 chooses the villager 82% of the time. This suggests that, in this situation, Agent-131 mostly chooses the leftmost object, likely because the right object blends into the background.

So we zoom out again. We repeat the same 100-rollout measurement across two object orderings, two environment seeds, and 100 agents. In seed 49, many agents exhibit a strong left bias, visible in Figure 5a. In the more symmetrical seed 481, many agents choose the regular tree more often regardless of where it is positioned, as shown in Figure 5b. And the behaviour is not merely a property of the environment: in seed 49, with villager-tree on the right and regular tree on the left, Agent-131 chooses the tree 98% of the time, while Agent-132 chooses the villager 74% of the time, as shown in Figure 6. After broader and broader experiments, varying object positions, backgrounds, and now whole agents, could we finally make claims about STEVE-1's behaviour? Our answer is still 'no': we could only make claims about the particular checkpoint or population we tested, not the STEVE-1 training procedure as a whole. Even wider experiments could include different prompts, hyperparameters, datasets, or even environments. Our model zoo confirms previous findings [15, 16] and shows why MultiSTEVE-1s is useful: training another agent using the same procedure, but different randomness (different seeds, GPUs, or libraries), can produce wildly different out-of-distribution behaviours. We should therefore study populations of agents whenever we can, rather than treating one checkpoint as the behaviour of the training procedure.

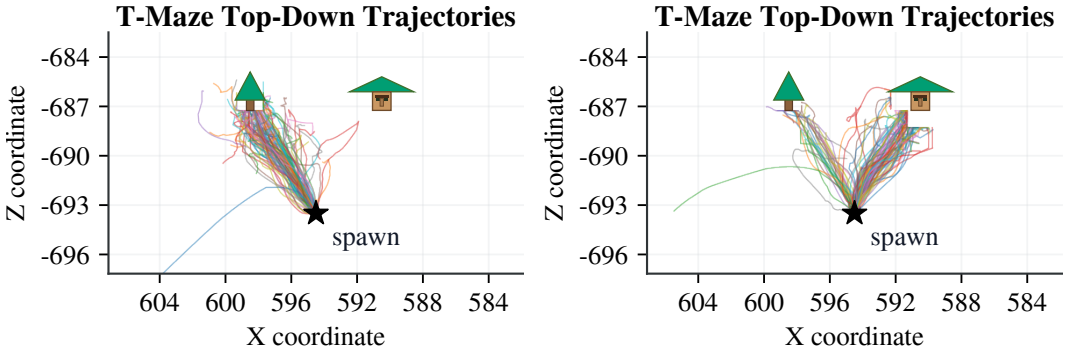
(a) Seed 49 (b) Seed 481 (c) Up-close villager view

Figure 4: (a), (b) First-frame examples of the T-maze task for two environment seeds. Seed 481 has a much more symmetrical background. (c) Up-close view of the villager-tree: clearly not a regular tree. All images show the downsampled agent view.



(a) Seed 49 (b) Seed 481

Figure 5: Agent choices in the T-maze experiment for all 100 agents in group 05, with the villager-tree and regular tree swapped between sides under two environment seeds. Seed 49 pulls agents towards the left regardless of the object ('always left' corner). The more symmetrical seed 481 shows agents preferring the tree ('regular tree' corner). See Figure 25 for the version with confidence intervals.



(a) Agent-131 (b) Agent-132

Figure 6: 100 top-down trajectories each of Agent-131 and Agent-132 in environment seed 49, with villager-tree on the right and regular tree on the left. In this scenario, Agent-131 nearly always chooses the regular tree, while Agent-132 strongly prefers the villager-tree.

Villager killing. The T-maze task shows that agents can differ in their choices, while the in-distribution evaluations show that they are relatively similar in their ability to find and chop trees. If there were only a single villager-tree in sight, would they differ in how often they kill the villager?

A villager has 200 health and takes around 27 punches to kill. We use environment seed 49 with a villager-tree on the left and no other trees in sight. To identify potential outliers in villager-killing behaviour, we perform 20 rollouts for each of the same 100 agents used in the T-maze task. After identifying Agent-111 and Agent-114 as two very different candidates, we perform 2,000 rollouts for each agent: 100 episodes for each combination of 10 environment seeds and villager-tree side. We choose the environment seeds such that the villager-tree is always the most prominent tree in view.

Agent-111 kills 70 villagers, while Agent-114 kills 908: 3.5% and 45.4%, respectively. See Figure 14 for full data. Qualitative inspection shows that both agents reliably approach the villager-tree, but Agent-111 usually throws only a few punches before looking up to hit the logs and leaves above, while Agent-114 continues punching the villager. Figure 4c shows the downsampled agent view of the nearby villager-tree: it is clearly not a tree. We also used the same procedure to identify outliers in the smaller group of ten agents in the bit-flip setting. We identified bit-flip agents 2 and 8 as the most different ones and found that they killed 31% and 51% of the villagers in 2,000 episodes each.

Can we therefore conclude that Agent-111 is less murderous than Agent-114? After all, we checked multiple environment seeds and villager-tree locations. We think you'll find it's a bit more complicated than that [48]. First, we checked only one of the 98 villager types in Minecraft [49], and only one of the many tree types. In addition, all of our environment seeds had the agent act in similar biomes. Most importantly, however, we tried two prompts when looking for outliers among the 100 agents: 'chop a tree', and the engineered prompt 'chop down the tree, gather wood, pick up wood, chop it down, break tree'. The second prompt increased the average villager-killing rate. We also noticed that agents often stopped hitting the villager to look up and start chopping the log above the villager's head. Removing this distracting log also increases the villager-killing rate. Of these four combinations of prompt and log settings, we chose the one with the lowest overall kill rate when performing an in-depth comparison of Agent-111 and Agent-114. This showed a 13-fold difference in kill rates: 3.5% and 45.4%. While in the other three settings the direction of the difference between these two agents stayed the same, it was not statistically significant in our initial sweeps with only 20 episodes per agent. There are clear differences in out-of-distribution behaviour between agents, but the differences are sensitive to many factors; see Figure 22 and Figure 23 in the appendix for full data for the 100 agents in all four settings. Thus, we can only claim that Agent-114 kills roughly 13 times more villagers than Agent-111 in that specific setting.

6 Conclusion

We introduce MultiSTEVE-1s, a model zoo and interpretability suite based on the instruction-following vision agent STEVE-1 [8]. We demonstrate how small randomness variations in the fine-tuning procedure can lead to very different rates of out-of-distribution goal misgeneralisation. These results suggest an inexpensive way to produce multiple model organisms of misalignment with meaningfully different behaviour. The main limitation is the narrowness of the zoo: it contains only one type of agent, with both dataset and hyperparameters fixed. This narrow focus reflects the state of the field: open-weight releases of similar large-scale agents are rare. We therefore see MultiSTEVE-1s primarily as research infrastructure for follow-up work in a relatively neglected part of the broader AI alignment ecosystem.

The intended impact of this release is to support alignment and interpretability research by enabling more robust comparisons across similarly trained agents. Direct misuse risk is limited because the agents act only in Minecraft through MineRL, without real-world actuation. The main negative considerations are capability-relevant infrastructure, compute cost, and asset provenance; we address these by documenting compute use and release provenance, and by framing the tools for analysis rather than deployment.

Acknowledgments and Disclosure of Funding

References

[1] S. P. Lehalleur, J. Hoogland, M. Farrugia-Roberts, S. Wei, A. G. Oldenziel, G. Wang, L. Carroll, and D. Murfet. You are what you eat – ai alignment requires understanding how data shapes structure and generalisation, 2025. URL https://arxiv.org/abs/2502.05475.

[2] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019. doi: 10.1038/s41586-019-1724-z.

[3] OpenAI, :, C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. d. O. Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang. Dota 2 with large scale deep reinforcement learning, 2019. URL https://arxiv.org/abs/1912.06680.

[4] S. Reed, K. Zolna, E. Parisotto, S. G. Colmenarejo, A. Novikov, G. Barth-maron, M. Giménez, Y. Sulsky, J. Kay, J. T. Springenberg, T. Eccles, J. Bruce, A. Razavi, A. Edwards, N. Heess, Y. Chen, R. Hadsell, O. Vinyals, M. Bordbar, and N. de Freitas. A generalist agent. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856. URL https://openreview.net/forum?id=1ikK0kHjvj. Featured Certification, Outstanding Certification.

[5] A. A. Team, J. Bauer, K. Baumli, S. Baveja, F. Behbahani, A. Bhoopchand, N. Bradley-Schmieg, M. Chang, N. Clay, A. Collister, V. Dasagi, L. Gonzalez, K. Gregor, E. Hughes, S. Kashem, M. Loks-Thompson, H. Openshaw, J. Parker-Holder, S. Pathak, N. Perez-Nieves, N. Rakicevic, T. Rocktäschel, Y. Schroecker, J. Sygnowski, K. Tuyls, S. York, A. Zacherl, and L. Zhang. Human-timescale adaptation in an open-ended task space, 2023. URL https://arxiv.org/abs/2301.07608.

[6] S. Team, M. A. Raad, A. Ahuja, C. Barros, F. Besse, A. Bolt, A. Bolton, B. Brownfield, G. Buttimore, M. Cant, S. Chakera, S. C. Y. Chan, J. Clune, A. Collister, V. Copeman, A. Cullum, I. Dasgupta, D. de Cesare, J. D. Trapani, Y. Donchev, E. Dunleavy, M. Engelcke, R. Faulkner, F. Garcia, C. Gbadamosi, Z. Gong, L. Gonzales, K. Gupta, K. Gregor, A. O. Hallingstad, T. Harley, S. Haves, F. Hill, E. Hirst, D. A. Hudson, J. Hudson, S. Hughes-Fitt, D. J. Rezende, M. Jasarevic, L. Kampis, R. Ke, T. Keck, J. Kim, O. Knagg, K. Kopparapu, R. Lawton, A. Lampinen, S. Legg, A. Lerchner, M. Limont, Y. Liu, M. Loks-Thompson, J. Marino, K. M. Cussons, L. Matthey, S. Mcloughlin, P. Mendolicchio, H. Merzic, A. Mitenkova, A. Moufarek, V. Oliveira, Y. Oliveira, H. Openshaw, R. Pan, A. Pappu, A. Platonov, O. Purkiss, D. Reichert, J. Reid, P. H. Richemond, T. Roberts, G. Ruscoe, J. S. Elias, T. Sandars, D. P. Sawyer, T. Scholtes, G. Simmons, D. Slater, H. Soyer, H. Strathmann, P. Stys, A. C. Tam, D. Teplyashin, T. Terzi, D. Vercelli, B. Vujatovic, M. Wainwright, J. X. Wang, Z. Wang, D. Wierstra, D. Williams, N. Wong, S. York, and N. Young. Scaling instructable agents across many simulated worlds, 2024. URL https://arxiv.org/abs/2404.10179.

[7] B. Baker, I. Akkaya, P. Zhokov, J. Huizinga, J. Tang, A. Ecoffet, B. Houghton, R. Sampedro, and J. Clune. Video pretraining (vpt): Learning to act by watching unlabeled online videos. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 24639–24654. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/9c7008aff45b5d8f0973b23e1a22ada0-Paper-Conference.pdf.

[8] S. Lifshitz, K. Paster, H. Chan, J. Ba, and S. A. McIlraith. STEVE-1: A generative model for text-to-behavior in minecraft. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=YkBDJWerKg.

[9] K. Jucys, G. Adamopoulos, M. Hamidi, S. Milani, M. R. Samsami, A. Zholus, S. Joseph, B. Richards, I. Rish, and Özgür Şimşek. Interpretability in action: Exploratory analysis of vpt, a minecraft agent, 2024. URL https://arxiv.org/abs/2407.12161.

[10] R. Shah, V. Varma, R. Kumar, M. Phuong, V. Krakovna, J. Uesato, and Z. Kenton. Goal misgeneralization: Why correct specifications aren't enough for correct goals, 2022. URL https://arxiv.org/abs/2210.01790.

[11] L. L. D. Langosco, J. Koch, L. D. Sharkey, J. Pfau, and D. Krueger. Goal misgeneralization in deep reinforcement learning. In K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 12004–12019. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/langosco22a.html.

[12] C. Summers and M. J. Dinneen. Nondeterminism and instability in neural network optimization. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 9913–9922. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/summers21a.html.

[13] P. Madhyastha and R. Jain. On model stability as a function of random seed, 2019. URL https://arxiv.org/abs/1909.10447.

[14] K. Jordan. On the variance of neural network training with respect to test sets and distributions. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=pEGSdJu52I.

[15] A. D'Amour, K. Heller, D. Moldovan, B. Adlam, B. Alipanahi, A. Beutel, C. Chen, J. Deaton, J. Eisenstein, M. D. Hoffman, F. Hormozdiari, N. Houlsby, S. Hou, G. Jerfel, A. Karthikesalingam, M. Lucic, Y. Ma, C. McLean, D. Mincu, A. Mitani, A. Montanari, Z. Nado, V. Natarajan, C. Nielson, T. F. Osborne, R. Raman, K. Ramasamy, R. Sayres, J. Schrouff, M. Seneviratne, S. Sequeira, H. Suresh, V. Veitch, M. Vladymyrov, X. Wang, K. Webster, S. Yadlowsky, T. Yun, X. Zhai, and D. Sculley. Underspecification presents challenges for credibility in modern machine learning. *Journal of Machine Learning Research*, 23 (226):1–61, 2022. URL http://jmlr.org/papers/v23/20-1335.html.

[16] T. Sellam, S. Yadlowsky, I. Tenney, J. Wei, N. Saphra, A. D'Amour, T. Linzen, J. Bastings, I. R. Turc, J. Eisenstein, D. Das, and E. Pavlick. The multiBERTs: BERT reproductions for robustness analysis. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=K0E_F0gFDgA.

[17] S. Biderman, H. Schoelkopf, Q. Anthony, H. Bradley, K. O'Brien, E. Hallahan, M. A. Khan, S. Purohit, U. S. Prashanth, E. Raff, A. Skowron, L. Sutawika, and O. Van Der Wal. Pythia: a suite for analyzing large language models across training and scaling. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

[18] O. van der Wal, P. Lesci, M. Müller-Eberstein, N. Saphra, H. Schoelkopf, W. Zuidema, and S. Biderman. Polypythias: Stability and outliers across fifty language model pre-training runs. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=bmrYu2Ekdz.

[19] S. Wei, D. Murfet, M. Gong, H. Li, J. Gell-Redman, and T. Quella. Deep learning is singular, and that's good. *IEEE Transactions on Neural Networks and Learning Systems*, 34(12):10473–10486, December 2023. ISSN 2162-2388. doi: 10.1109/tnnls.2022.3167409. URL http://dx.doi.org/10.1109/TNNLS.2022.3167409.

[20] C. Elliott, E. Urdshals, D. Quarel, M. Farrugia-Roberts, and D. Murfet. Stagewise reinforcement learning and the geometry of the regret landscape, 2026. URL https://arxiv.org/abs/2601.07524.

[21] H. Petzka, M. Kamp, L. Adilova, C. Sminchisescu, and M. Boley. Relative flatness and generalization. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 18420–18432. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/995f5e03890b029865f402e83a81c29d-Paper.pdf.

[22] S. Hong, P. Frigo, Y. Kaya, C. Giuffrida, and T. Dumitraş. Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks, 2019. URL https://arxiv.org/abs/1906.01017.

[23] L. Gao, J. Schulman, and J. Hilton. Scaling laws for reward model overoptimization. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

[24] J. Clark and D. Amodei. Faulty reward functions in the wild. https://openai.com/index/faulty-reward-functions/, December 2016. Blog post.

[25] K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman. Quantifying generalization in reinforcement learning. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1282–1289. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/cobbe19a.html.

[26] Y. Jiang, B. Neyshabur, H. Mobahi, D. Krishnan, and S. Bengio. Fantastic generalization measures and where to find them. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SJgIPJBFvH.

[27] S. Zhang, I. Reid, G. V. Pérez, and A. Louis. Why flatness does and does not correlate with generalization for deep neural networks, 2021. URL https://arxiv.org/abs/2103.06219.

[28] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=Sy8gdB9xx.

[29] B. Neyshabur, S. Bhojanapalli, D. McAllester, and N. Srebro. Exploring generalization in deep learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 5949–5958, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

[30] S. Hochreiter and J. Schmidhuber. Flat minima. *Neural Comput.*, 9(1):1–42, January 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.1.1. URL https://doi.org/10.1162/neco.1997.9.1.1.

[31] C. Olsson, N. Elhage, N. Nanda, N. Joseph, N. DasSarma, T. Henighan, B. Mann, A. Askell, Y. Bai, A. Chen, T. Conerly, D. Drain, D. Ganguli, Z. Hatfield-Dodds, D. Hernandez, S. Johnston, A. Jones, J. Kernion, L. Lovitt, K. Ndousse, D. Amodei, T. Brown, J. Clark, J. Kaplan, S. McCandlish, and C. Olah. In-context learning and induction heads, 2022. URL https://arxiv.org/abs/2209.11895.

[32] M. Hanna, O. Liu, and A. Variengien. How does gpt-2 compute greater-than?: Interpreting mathematical abilities in a pre-trained language model, 2023. URL https://arxiv.org/abs/2305.00586.

[33] A. Karvonen, C. Rager, J. Lin, C. Tigges, J. I. Bloom, D. Chanin, Y.-T. Lau, E. Farrell, C. S. McDougall, K. Ayonrinde, D. Till, M. Wearden, A. Conmy, S. Marks, and N. Nanda. SAEBench: A comprehensive benchmark for sparse autoencoders in language model interpretability. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=qrU3yNfX0d.

[34] E. Voita, D. Talbot, F. Moiseev, R. Sennrich, and I. Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned, 2019. URL https://arxiv.org/abs/1905.09418.

[35] K. Clark, U. Khandelwal, O. Levy, and C. D. Manning. What does bert look at? an analysis of bert's attention, 2019. URL https://arxiv.org/abs/1906.04341.

[36] U. Mini, P. Grietzer, M. Sharma, A. Meek, M. MacDiarmid, and A. M. Turner. Understanding and controlling a maze-solving policy network, 2023. URL https://arxiv.org/abs/2310.08043.

[37] L. Sharkey, S. Black, and Beren. Current themes in mechanistic interpretability research. https://www.lesswrong.com/posts/Jgs7LQwmvErxR9BCC/current-themes-in-mechanistic-interpretability-research, November 2022. LessWrong.

[38] E. Hubinger, C. Denison, J. Mu, M. Lambert, M. Tong, M. MacDiarmid, T. Lanham, D. M. Ziegler, T. Maxwell, N. Cheng, A. Jermyn, A. Askell, A. Radhakrishnan, C. Anil, D. Duvenaud, D. Ganguli, F. Barez, J. Clark, K. Ndousse, K. Sachan, M. Sellitto, M. Sharma, N. DasSarma, R. Grosse, S. Kravec, Y. Bai, Z. Witten, M. Favaro, J. Brauner, H. Karnofsky, P. Christiano, S. R. Bowman, L. Graham, J. Kaplan, S. Mindermann, R. Greenblatt, B. Shlegeris, N. Schiefer, and E. Perez. Sleeper agents: Training deceptive llms that persist through safety training, 2024. URL https://arxiv.org/abs/2401.05566.

[39] J. Hoogland, A. G. Oldenziel, D. Murfet, and S. van Wingerden. Towards developmental interpretability. In *AI Alignment Forum*, 2023.

[40] L. Fan, G. Wang, Y. Jiang, A. Mandlekar, Y. Yang, H. Zhu, A. Tang, D.-A. Huang, Y. Zhu, and A. Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022. URL https://openreview.net/forum?id=rc8o_j8I8PX.

[41] W. H. Guss, B. Houghton, N. Topin, P. Wang, C. Codel, M. Veloso, and R. Salakhutdinov. Minerl: a large-scale dataset of minecraft demonstrations. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, IJCAI'19, page 2442–2448. AAAI Press, 2019. ISBN 9780999241141.

[42] M. Sasdelli, T. Ajanthan, T.-J. Chin, and G. Carneiro. A chaos theory approach to understand neural network optimization. In *2021 Digital Image Computing: Techniques and Applications (DICTA)*, pages 1–10, 2021. doi: 10.1109/DICTA52665.2021.9647143.

[43] S. Joseph, A. Zholus, M. R. Samsami, and B. A. Richards. Mining the diamond miner: Mechanistic interpretability on the video pretraining agent, 2023. URL https://openreview.net/pdf?id=lDeysxpH6W. Workshop on Attributing Model Behavior at Scale (ATTRIB) at NeurIPS 2023.

[44] V. Krakovna, J. Uesato, V. Mikulik, M. Rahtz, T. Everitt, R. Kumar, Z. Kenton, J. Leike, and S. Legg. Specification gaming: the flip side of AI ingenuity. https://deepmind.google/blog/specification-gaming-the-flip-side-of-ai-ingenuity/, April 2020. DeepMind Blog.

[45] D. S. Olton. Mazes, maps, and memory. *American Psychologist*, 34(7):583–596, 1979. doi: 10.1037//0003-066x.34.7.583.

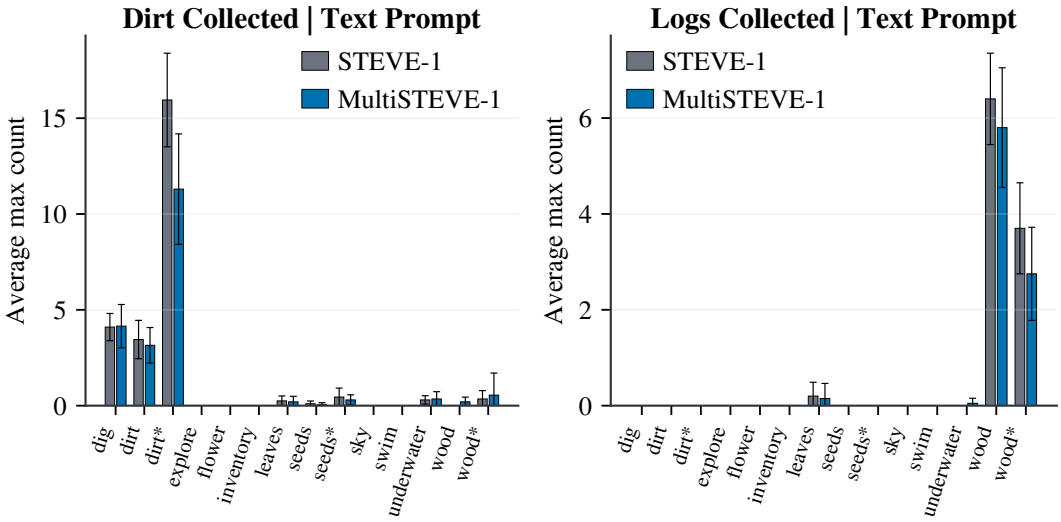
[46] R. d'Isa, G. Comi, and L. Leocani. Apparatus design and behavioural testing protocol for the evaluation of spatial working memory in mice through the spontaneous alternation T-maze. *Scientific Reports*, 11(1): 21177, 2021. doi: 10.1038/s41598-021-00402-7.

[47] H. Shoji, H. Hagihara, K. Takao, S. Hattori, and T. Miyakawa. T-maze forced alternation and left-right discrimination tasks for assessing working and reference memory in mice. *Journal of Visualized Experiments*, page 3300, 2012. doi: 10.3791/3300.

[48] B. Goldacre. *I Think You'll Find It's a Bit More Complicated Than That*. Fourth Estate, London, 2015.

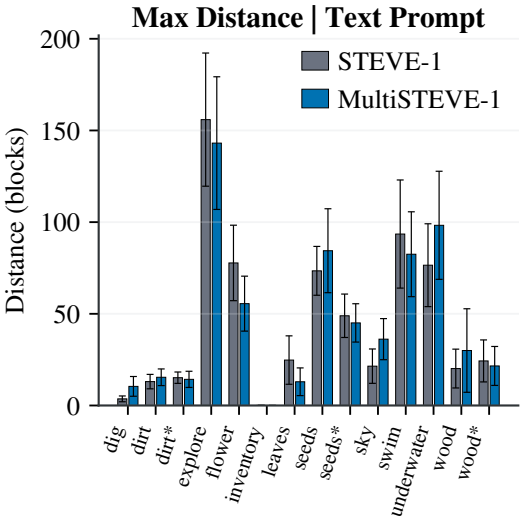
[49] Minecraft Wiki contributors. Java edition 1.16. https://minecraft.wiki/w/Java_Edition_1.16, 2020. Accessed: 2026-05-03.

A Full evaluations

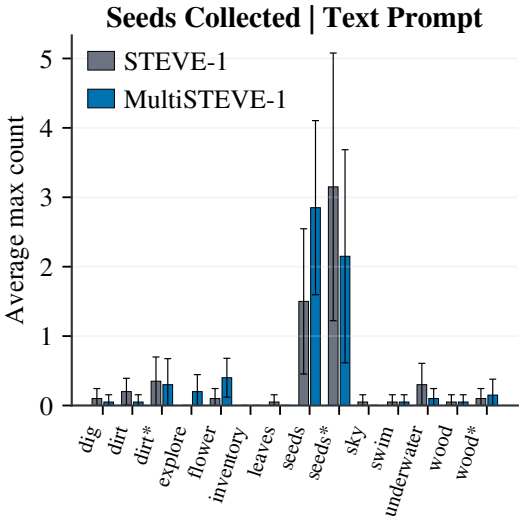


(a) Dirt

(b) Logs

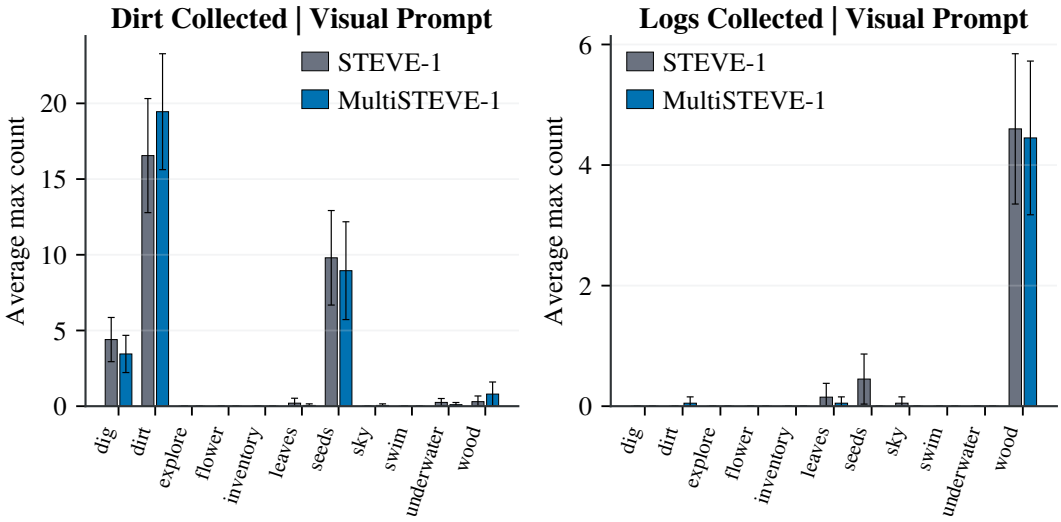


(c) Max distance



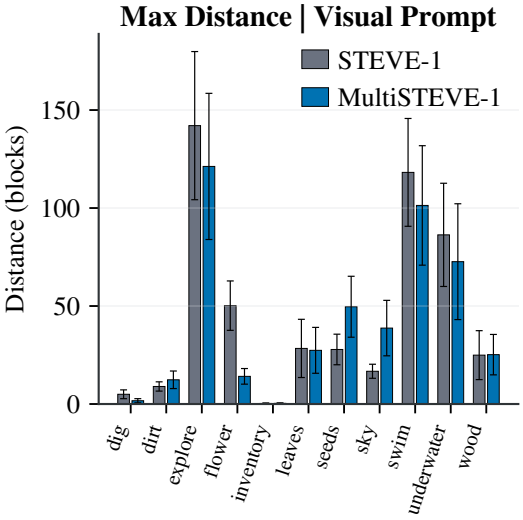
(d) Seeds

Figure 7: Results for text prompts.

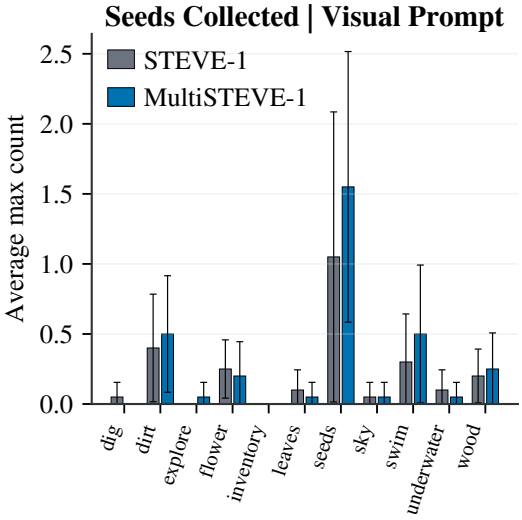


(a) Dirt

(b) Logs



(c) Max distance



(d) Seeds

Figure 8: Results for visual prompts.

B Partial evaluations

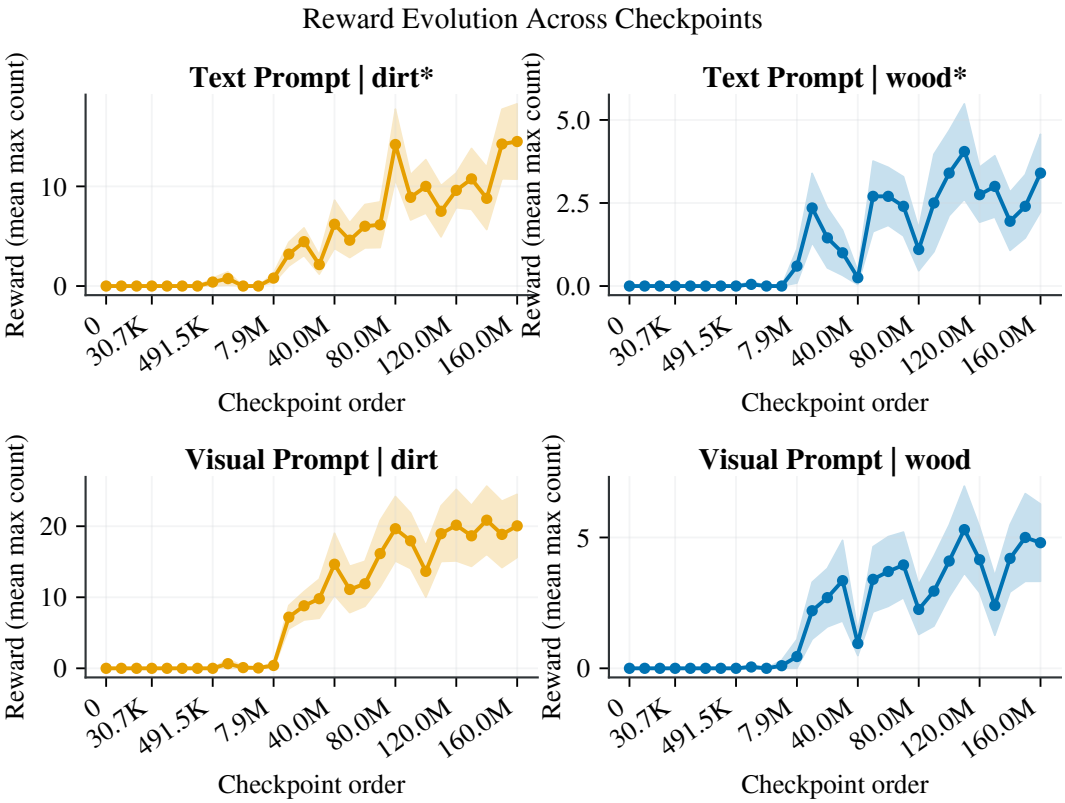


Figure 9: Performance during training.

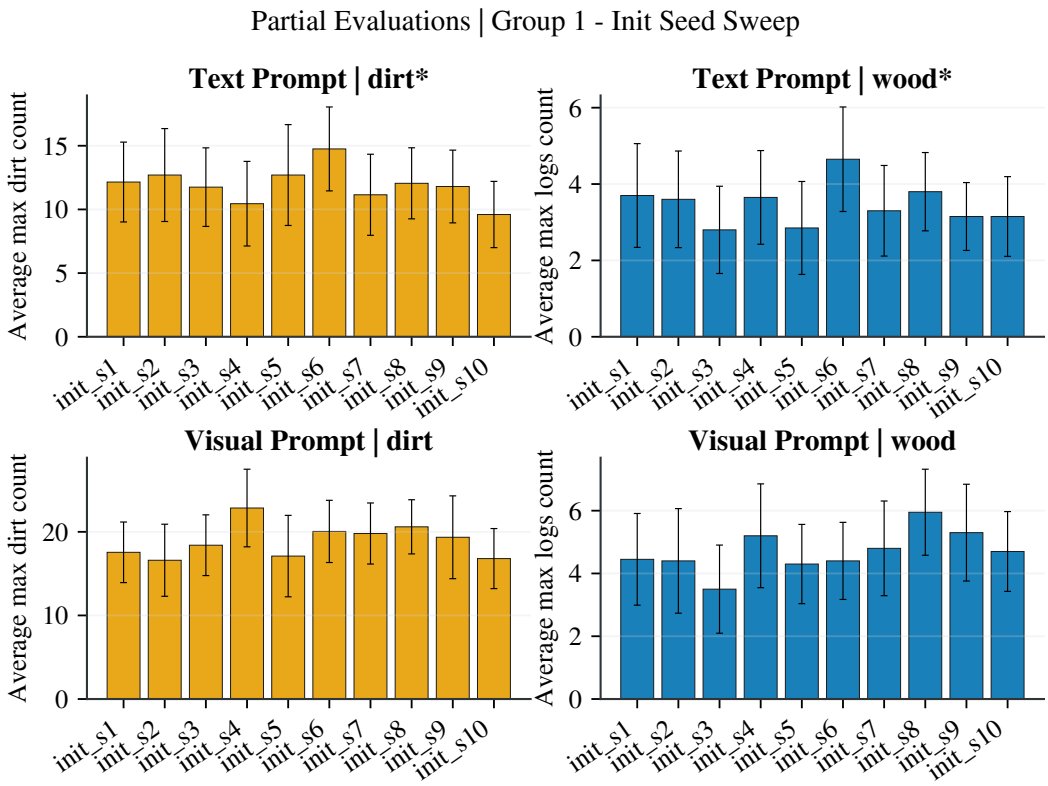


Figure 10: Group 01 partial evaluations.

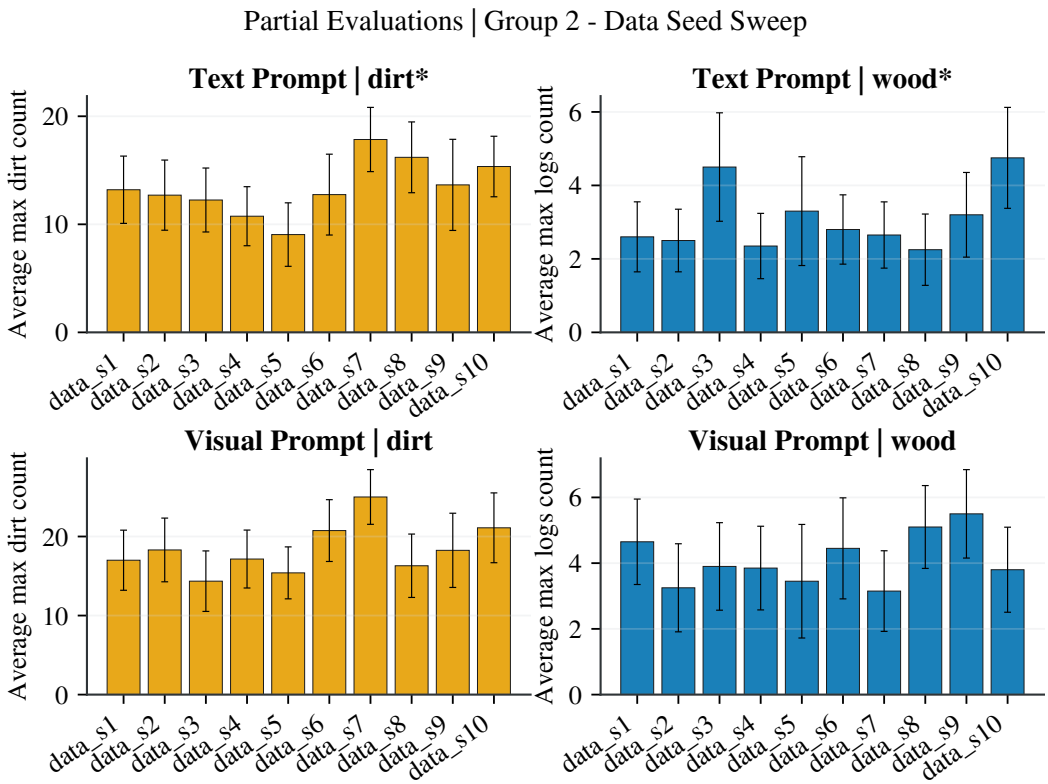


Figure 11: Group 02 partial evaluations.

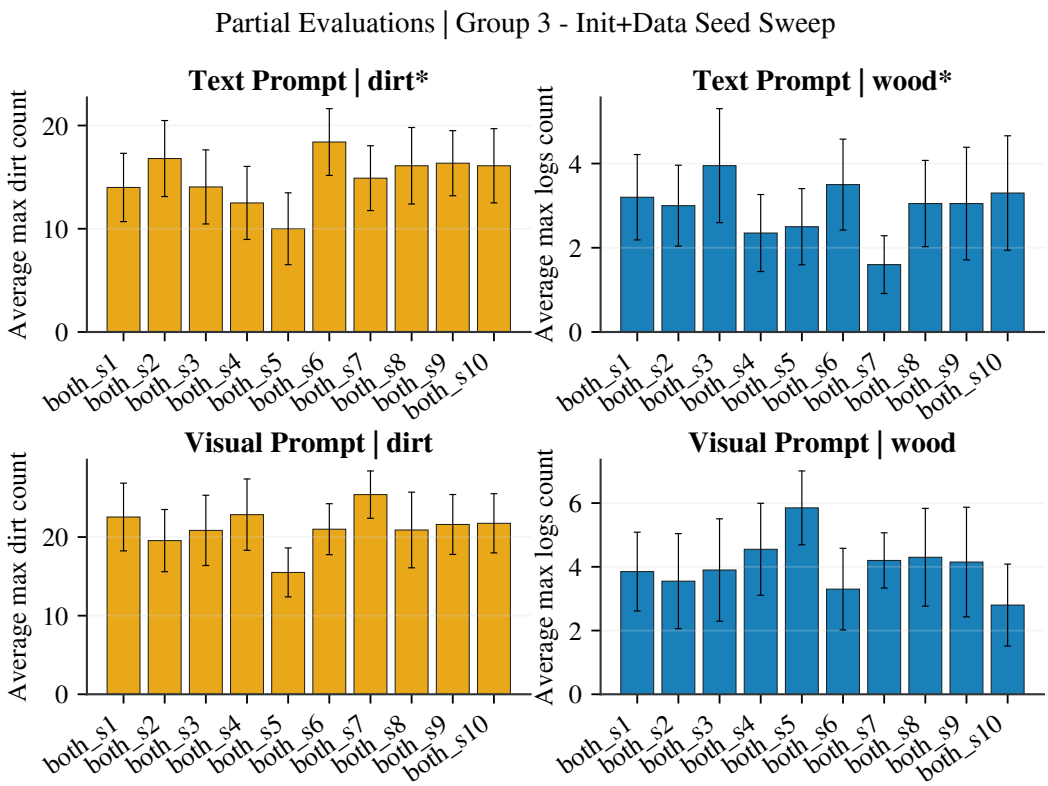


Figure 12: Group 03 partial evaluations.

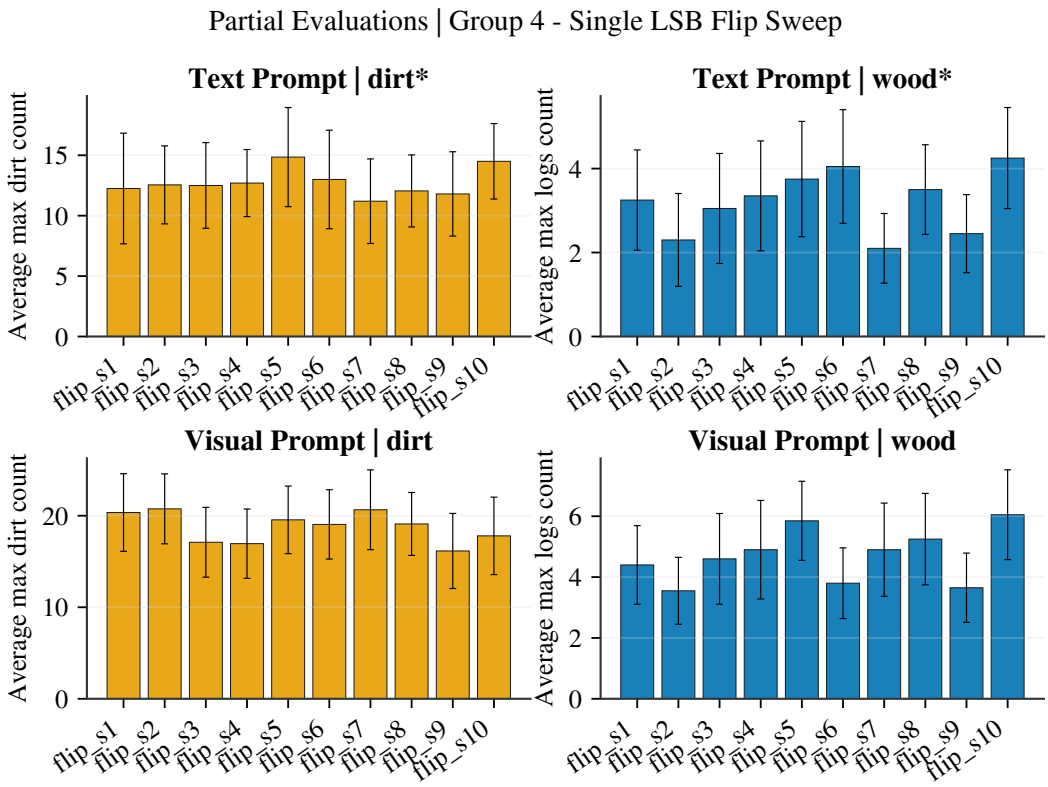


Figure 13: Group 04 partial evaluations.

C Villager chop

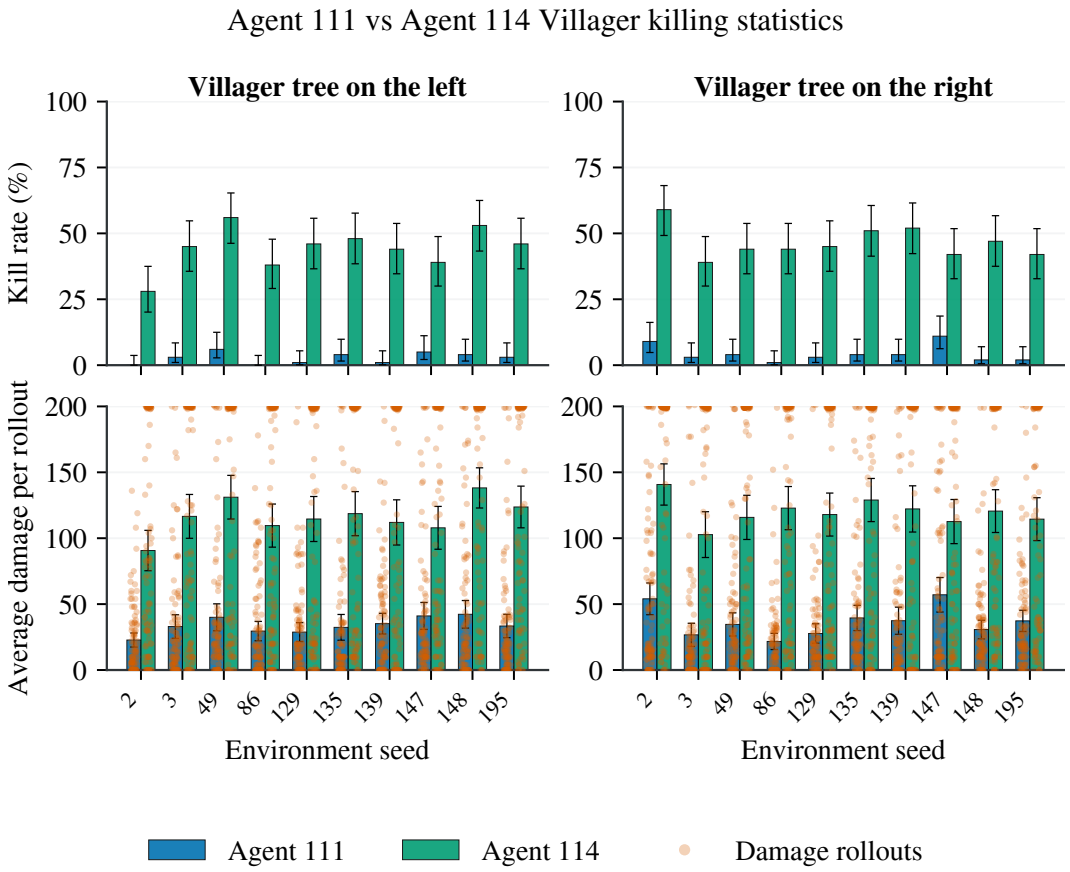


Figure 14: Agent-114 kills many more villagers than Agent-111 in the setting with the prompt 'chop a tree' and a log above the villager's head. This persists over 10 different environment seeds and regardless of which side the villager-tree is placed on.

D Attention patterns

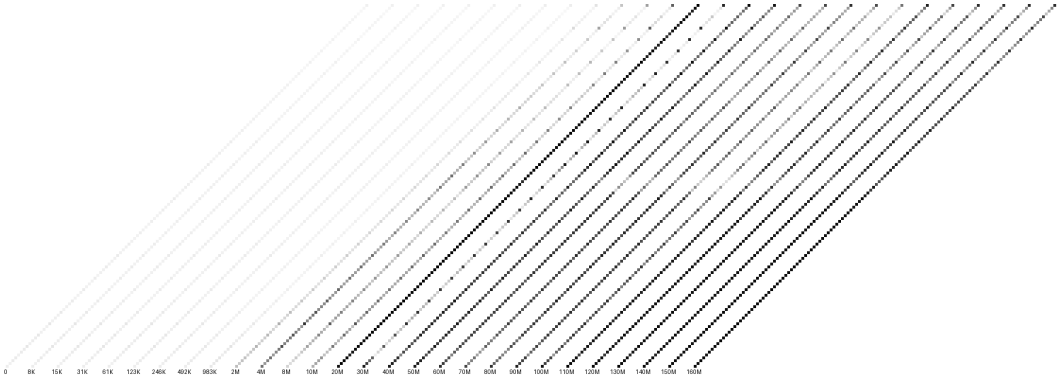


Figure 15: Attention paid to the last frame with open inventory by 28 model checkpoints of Agent-1 during training in a specific rollout.

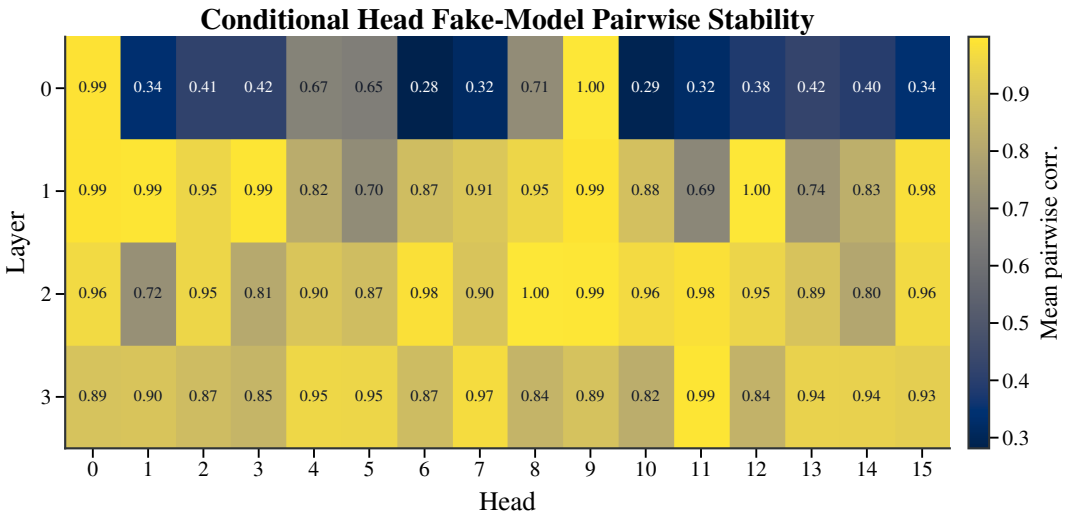


Figure 16: When the same rollout is shown to multiple agents, their attention-head patterns match more strongly in later layers. Some attention heads, such as 0.0, 0.9, and 2.1, are outliers under this metric.

E Randomness control

Although our least-significant-bit-flip result from subsection 5.1 is not new, it is surprising [12]. We have witnessed prominent researchers dismiss the original bit-flip result by saying 'that can't be true, surely they simply didn't control the randomness fully and that's why the bit-flip looks as different as full randomness.' Here we detail the steps we took to account for randomness and verify the bit-flip result.

Finding all sources of randomness. We found three sources of randomness in the original STEVE-1 training procedure: weight initialisation, data order, and number of frames between goals in the data. We add one seeded random number generator to control weight initialisation and another to control data order and goal frames. After fixing the two seeds, we vary the global random seeds for Python libraries such as torch, numpy, and random. The resulting training runs are bitwise identical, showing that all sources of randomness within the training code have been controlled. Note that this still requires fixed library versions and the same kind of GPU for all training runs. Our training runs are not bitwise identical across GPU models because matrix multiplications may be computed in different orders, and these operations are non-associative ($(a + b) + c$ may not be bitwise equal to $a + (b + c)$). We also found differences between CUDA library versions on some newer GPUs: older library versions did not fully support them and silently defaulted to less deterministic computational procedures.

Bitwise identical training runs. If we use the same random seeds, we should get bitwise identical final trained model weights. We confirm this with three separate but otherwise identical training runs using seeds 1 and 1, which appear in groups 01, 02, and 03 of our model zoo. Not only are the final trained models identical, so are all the training metrics using full numerical precision, except, of course, for the ones that use time as part of the calculation. In addition, we used three different compute clusters for training. On each cluster, we used the same GPU models and reran the (1, 1) model to confirm that the runs remained bitwise identical.

Extensive logging and initial checkpoints. We provide full anonymised training logs with full numerical precision in text format for each of the 140 models in our model zoo. We also provide logs with the exact data order used for training, saving the video name and starting and ending frame numbers for each 640-frame training segment in simple text format. The data order logs are identical for runs with identical data seeds, and differ for runs with different data seeds, as expected. We also save initial checkpoints for all 140 models before any training happens. When weight initialisation seeds are the same, these checkpoints are bitwise identical; when they differ, the checkpoints differ only in the MineCLIP layer, as expected. The initial checkpoints of bit-flip models differ from the parent (1, 1) model in only a single weight, and from each other in two weights: a random weight is flipped in each model, leading to two differences, as expected. One thing we failed to log was the random goal frames for each segment. However, because the goal frames are reproducible and fully determined by the data seed, this is a logging issue rather than a reproducibility issue. This is a minor limitation in our proof of full randomness control.

Resuming training runs. At specific intervals during training, we save resumable model checkpoints, which include model weights, optimizer states, dataloader states, and random number generator states. This should allow resumed training runs to be bitwise identical to uninterrupted training runs. However, some resumed training runs were not bitwise identical, seemingly at random. This happened despite identical random seeds and hyperparameters. After many iterations and many attempts to fix this, we decided not to use the resume functionality in our model zoo. Each training run was performed in a single session, and any run interrupted for any reason was discarded and restarted from scratch. This is a limitation of our work and prevents some types of research into training dynamics, such as resuming multiple times from an earlier checkpoint and continuing training in different ways.

Evaluation randomness However, MineRL rollouts are not deterministic even when fully seeded, so any evaluations and experiments involving rollouts are also not deterministic. This comes from the MineRL environment not fully controlling the randomness of certain elements of the Minecraft world, such as mob spawning and colours of some flowers. In addition, world chunk loading order and speed can be random and depends on how busy the CPU that runs the world is.

F Which bit is the least significant?

Before TF32 support was introduced in PyTorch 1.7, the answer was straightforward: the least significant bit was the last bit in the binary representation of an FP32 parameter, that is, the last bit of the mantissa. This corresponds to the smallest possible change in a floating point number and matches the bit-flip setup used in prior nondeterminism experiments with PyTorch 1.3.1 [12]. With the later introduction of TF32 and Ampere GPU architectures, some matrix multiplication and convolution operations during neural network training used a faster reduced-precision mode, rounding inputs to TF32 precision with 10 mantissa bits instead of FP32's 23 stored mantissa bits. Thus, changing the very last bit of the FP32 number would only affect some computations, while having no effect on TF32-rounded ones. We observed that this sometimes led to differences in final trained models, while other times they converged to the same bitwise representation due to this rounding. We decided to use the 14th bit from the end, that is, the least significant bit that TF32 operations do not discard, which is the smallest possible change in a floating point number that survives TF32 rounding in matrix multiplication and convolution operations.

G Differences compared to the original STEVE-1

Below is a list of confirmed and potential differences between our training procedure and the original STEVE-1 training procedure.

- Type of GPUs: we used NVIDIA L40S instead of NVIDIA A40 GPUs. Both have 48 GB of VRAM, so we could use the same batch size of 12.
- Number of GPUs: we used one GPU per training run instead of four for reproducibility and randomness control.
- Learning rate: more GPUs often means a higher effective batch size and more stable gradients, allowing higher learning rates. We experimented with lower learning rates, but kept the original value of 4e-5.
- gradient_accumulation_steps: this hyperparameter is the only one not specified in the original paper. Different parts of the codebase use different defaults, 4 and 1. We could not clarify which value the authors used, so we chose 4 because it gives more stable gradient updates.
- Potential gradient reset bug: if the original authors used 4 for gradient_accumulation_steps, they may have discarded 75% of gradients because of a mistimed gradient reset during accumulation. We fix this bug in our code.
- Randomness control: as discussed in Appendix E, we implement separate local seeds for each source of randomness instead of depending on global seeds.

H Python Library Versions

The following Python packages were used during training.

Package	Version	Package	Version	Package	Version
absl-py	2.3.1	imageio	2.37.0	ptyprocess	0.7.0
accelerate	0.18.0	imageio-ffmpeg	0.3.0	pure-eval	0.2.3
antlr4-python3-runtime	4.9.3	importlib-resources	6.5.2	pycparser	2.23
appdirs	1.4.4	inflection	0.5.1	pyglet	2.1.8
asttokens	3.0.1	iniconfig	2.3.0	pygments	2.19.2
attrs	25.3.0	ipython	8.38.0	pyparsing	3.3.2
beautifulsoup4	4.13.5	jedi	0.19.2	Pyro4	4.82
brotlicffi	1.0.9.2	jinja2	3.1.6	PySocks	1.7.1
bullet	2.2.0	jsonlines	4.0.0	pytest	9.0.2
certifi	2026.1.4	kiwisolver	1.4.9	python-dateutil	2.9.0.post0
cffi	1.17.1	kornia	0.8.1	pytorch-warmup	0.2.0
charset-normalizer	3.4.4	kornia-rs	0.1.9	pytz	2025.2
click	8.3.0	lxml	6.0.2	pywavelets	1.8.0
cloudpickle	1.6.0	MarkupSafe	2.1.5	PyYAML	6.0.2
coloredlogs	15.0.1	matplotlib	3.10.8	regex	2025.9.18
contourpy	1.3.2	matplotlib-inline	0.2.1	requests	2.32.5
cycler	0.12.1	mineclip	0.1	safetensors	0.6.2
daemoniker	0.2.3	minedojo	0.1	scipy	1.15.3
decorator	5.2.1	minerl	1.0.0	seaborn	0.13.2
dill	0.4.1	mkl-fft	1.3.11	sentry-sdk	2.38.0
dm-tree	0.1.9	mkl-random	1.2.8	serpent	1.42
docker-pycreds	0.4.0	mkl-service	2.4.0	setproctitle	1.3.7
einops	0.8.1	moderngl	5.12.0	setuptools	80.10.2
entmax	1.3	mpmath	1.3.0	simple-term-menu	1.6.6
exceptiongroup	1.3.1	multiprocess	0.70.18	six	1.17.0
executing	2.2.1	mypy-extensions	1.1.0	smmap	5.0.2
filelock	3.20.0	networkx	3.4.2	soupsieve	2.8
flaky	3.8.1	numpy	2.2.6	stack-data	0.6.3
fonttools	4.61.1	nvidia-cublas-cu12	12.1.3.1	steve1	0.0.1
fsspec	2025.12.0	nvidia-cuda-cupti-cu12	12.1.105	sympy	1.13.1
gdown	5.2.0	nvidia-cuda-nvrtc-cu12	12.1.105	tokenizers	0.22.1
getch	1.0	nvidia-cuda-runtime-cu12	12.1.105	tomli	2.4.0
gitdb	4.0.12	nvidia-cudnn-cu12	9.1.0.70	torch	2.5.1+cu121
gitpython	3.1.45	nvidia-cufft-cu12	11.0.2.54	torchaudio	2.5.1
glcontext	3.0.0	nvidia-curand-cu12	10.3.2.106	torchvision	0.20.1+cu121
glfw	1.12.0	nvidia-cusolver-cu12	11.4.5.107	tqdm	4.67.3
gmpy2	2.2.1	nvidia-cusparse-cu12	12.1.0.106	traitlets	5.14.3
gym	0.19.0	nvidia-nccl-cu12	2.21.5	transformers	4.56.2
gym3	0.3.3	nvidia-nvjitlink-cu12	12.9.86	triton	3.1.0
hf-xet	1.1.10	nvidia-nvtx-cu12	12.1.105	typing	3.7.4.3
huggingface-hub	0.35.0	omegaconf	2.3.0	typing-extensions	4.15.0
humanfriendly	10.0	opencv-python	4.11.0.86	tzdata	2025.2
hydra-core	1.3.2	opencv-python-headless	4.11.0.86	update-checker	0.18.0
idna	3.11	packaging	26.0	urllib3	2.6.3
ImageHash	4.3.2	pandas	2.3.2	wandb	0.15.12
imageio	2.37.0	parso	0.8.5	warmup-scheduler-pytorch	0.1.2
imageio-ffmpeg	0.3.0	pathtools	0.1.2	wcwidth	0.5.3
importlib-resources	6.5.2	pexpect	4.9.0	websocket-client	1.8.0
inflection	0.5.1	pillow	12.0.0	wget	3.2
iniconfig	2.3.0	pip	21.0	wheel	0.38.0
ipython	8.38.0	pluggy	1.6.0	wrapt	1.17.3
jedi	0.19.2	praw	7.8.1	x-transformers	0.27.1
jinja2	3.1.6	prawcore	2.4.0	xmltodict	0.12.0
jsonlines	4.0.0	prompt-toolkit	3.0.52		
kiwisolver	1.4.9	protobuf	4.25.8		
kornia	0.8.1	psutil	7.2.2		

I Computational resources and environmental impact.

Training. Although the baseline VPT agent took close to 170,000 GPU hours to train [7], fine-tuning it to produce a single STEVE-1 agent took 50 hours on one NVIDIA L40S GPU with 48 GB of VRAM, two CPUs, and 32 GB of RAM. Each training run with intermediate checkpoints takes about 30 GB of disc space. The full model zoo size is 1.3 TB. The size of the embedded datasets is 352 GB. Training 140 agents took 7,000 GPU hours. Including early experiments and crashed or accidentally killed runs, we used around 10,000 GPU hours. We observed an average GPU power draw of 310 W during training, equivalent to 3,100 kWh of electricity. This is only direct GPU electricity consumption. Including host-system power and datacentre overhead, total electricity use would still stay in the low-single-digit MWh range. Based on the average CO_2 intensity of the locations of our compute clusters, we estimate 1–2 tonnes of total CO_2 emissions, roughly equal to a return London-New York flight for one passenger.

Evaluations. Evaluating the agents took roughly 1,000 GPU hours on a desktop PC with an NVIDIA GeForce RTX 4090 GPU with 24 GB of VRAM. Rollout generation takes only 3.5 GB of VRAM, three CPUs, and negligible RAM. However, rollout generation does not scale well beyond two parallel threads on a single machine.

Training and rollout generation can be done locally or on compute clusters, although rollouts are easier to run on local machines with a monitor. The CO_2 emissions from the evaluations and the 550 GPU hours used for data collection and embedding do not materially change the estimate of 1–2 tonnes of total CO_2 emissions for the project.

J Hyperparameters

Table 2: Training hyperparameters used for our STEVE-1 fine-tuning runs.

Hyperparameter	Value
Base model	`data/weights/vpt/2x.model`
Initial weights	`rl-from-foundation-2x`
Truncation length	64
Sequence length	640
Batch size	12
Gradient accumulation steps	4
Number of workers	1
Weight decay	0.039428
Training frames	160,000,000
Learning rate	4×10^{-5}
Warmup frames	10,000,000
Unconditional probability	0.1
Minimum frames between goals	15
Maximum frames between goals	200
Validation frequency	10,000
Initial validation frequency	10,000
Validation frequency switch steps	500
Shareable checkpoint frequency	10,000,000 frames
Resumable checkpoint frequency	20,000,000 frames
Validation sampling frequency	Every validation
Save each validation	False
Sampling strategy	`neurips`

K Text prompts

Table 3: Text prompts used in the paper. The starred labels denote engineered prompt variants.

Paper label	Eval slug	Exact text prompt
`dig`	`dig`	'dig as far as possible'
`dirt`	`dirt`	'get dirt'
`dirt*`	`dirt-engineered`	'get dirt, dig hole, dig dirt, gather a ton of dirt, collect dirt'
`sky`	`sky`	'look at the sky'
`leaves`	`leaves`	'break leaves'
`wood`	`wood`	'chop a tree'
`wood*`	`wood-engineered`	'chop down the tree, gather wood, pick up wood, chop it down, break tree'
`seeds`	`seeds`	'collect seeds'
`seeds*`	`seeds-engineered`	'break tall grass, break grass, collect seeds, punch the ground, run around in circles getting seeds from bushes'
`flower`	`flower`	'break a flower'
`explore`	`explore`	'go explore'
`swim`	`swim`	'go swimming'
`underwater`	`underwater`	'go underwater'
`inventory`	`inventory`	'open inventory'

L Interpretability suite resources

Below we show examples of the interpretability suite functionality and provide some additional resources to help with experiments.

L.1 Attention patterns dashboard

The attention-pattern dashboard lets users choose a specific episode rollout, select one of 64 attention heads to visualise, including combined views, and add multiple metrics from a list of over 100. Hovering over a specific attention-position and frame combination shows the corresponding values and the current and attended-to frames. Diagonal lines seen in Figure 17 represent attention being paid to a single frame in the past. Horizontal lines, such as the one in head 0.9 (see the online demo: https://multisteve-1s.netlify.app/), mean that attention is mostly paid to the frame in a specific memory position, for example the previous frame. Horizontal lines every 4 frames (see head 1.2 in the online demo) likely occur because of the 4-frame attack animation.

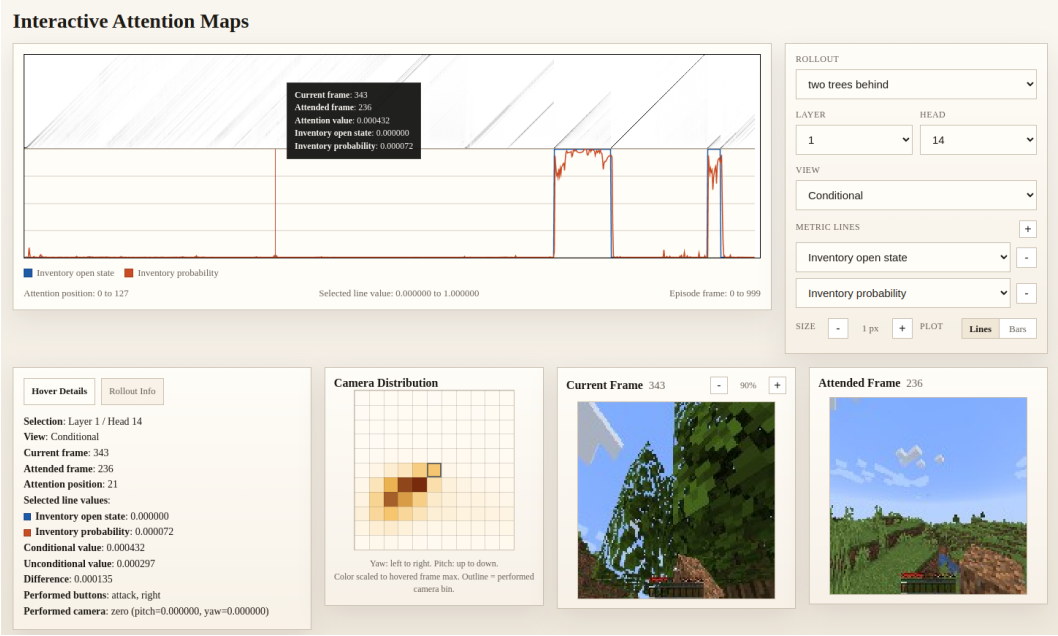


Figure 17: Attention patterns dashboard.

L.2 3D environment and trajectory viewer

The 3D environment viewer lets users inspect a specific Minecraft world from different angles and zoom levels, and view agent trajectories as they develop. See Figure 18 for the 3D trajectories of one of the T-maze experiments.

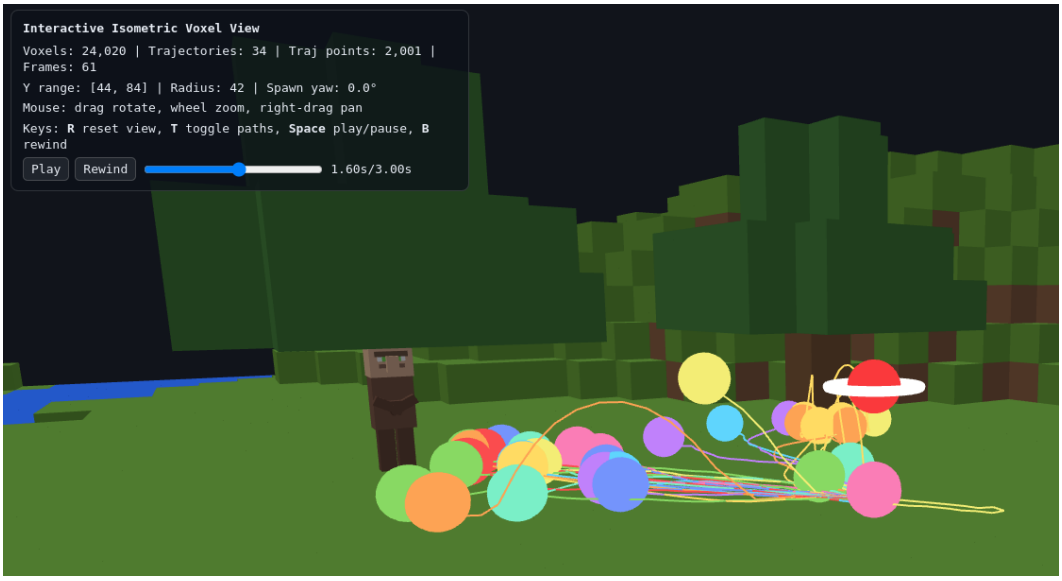


Figure 18: Interactive 3D voxel view of the MineRL environment with visualised agent trajectories. Each line is a trajectory with a sphere at the end representing the current agent position.

L.3 Interactive T-maze scatter plots

The interactive T-maze scatter plot shows the top-down trajectories and first frames for each agent. For example, Figure 19 shows the trajectories of Agent-141, the most regular-tree-seeking agent.

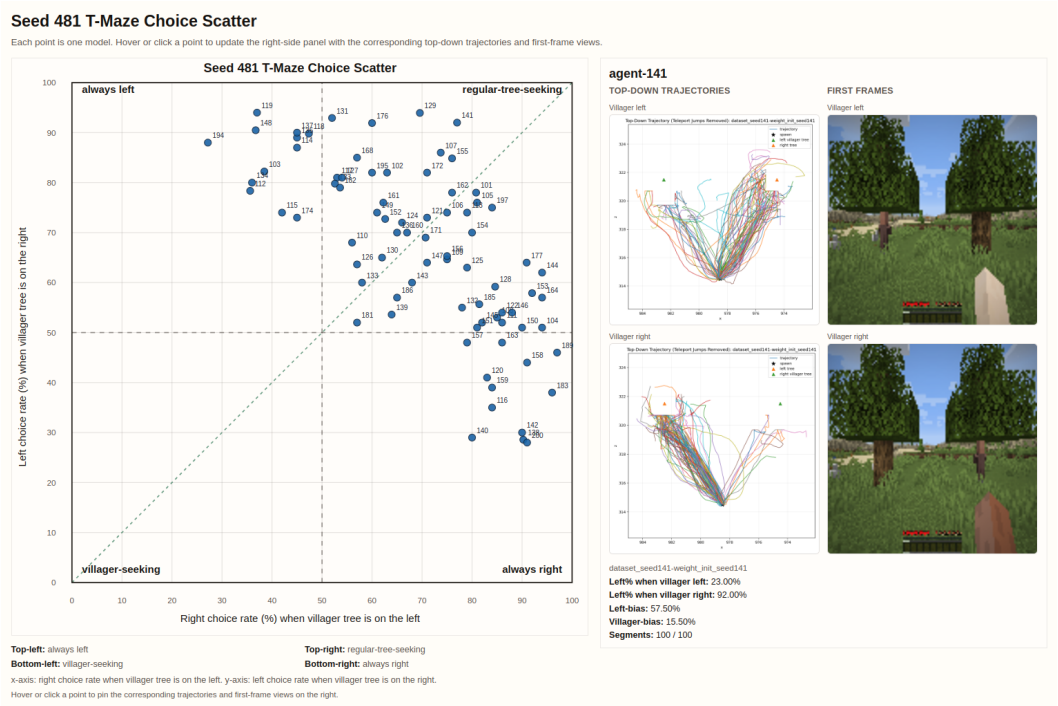


Figure 19: Interactive T-maze result explorer.

L.4 Environment seeds

You can find the first-observation images of over 17,000 environment seeds here: seeds. A nearly random sample of seeds can be seen in Figure 20. A sample of seeds with nearby villages, to elicit more morally questionable behaviour: 8, 449, 569, 686, 714, 1098, 6029, 6058, 6784, 16894. Seeds with clear flat ground in front of the agent for T-maze experiments: 2, 3, 49, 86, 129, 135, 139, 147, 148, 195, 273, 335, 359, 481, 522, 601, 603, 605, 668, 716, 736, 759, 794, 796.

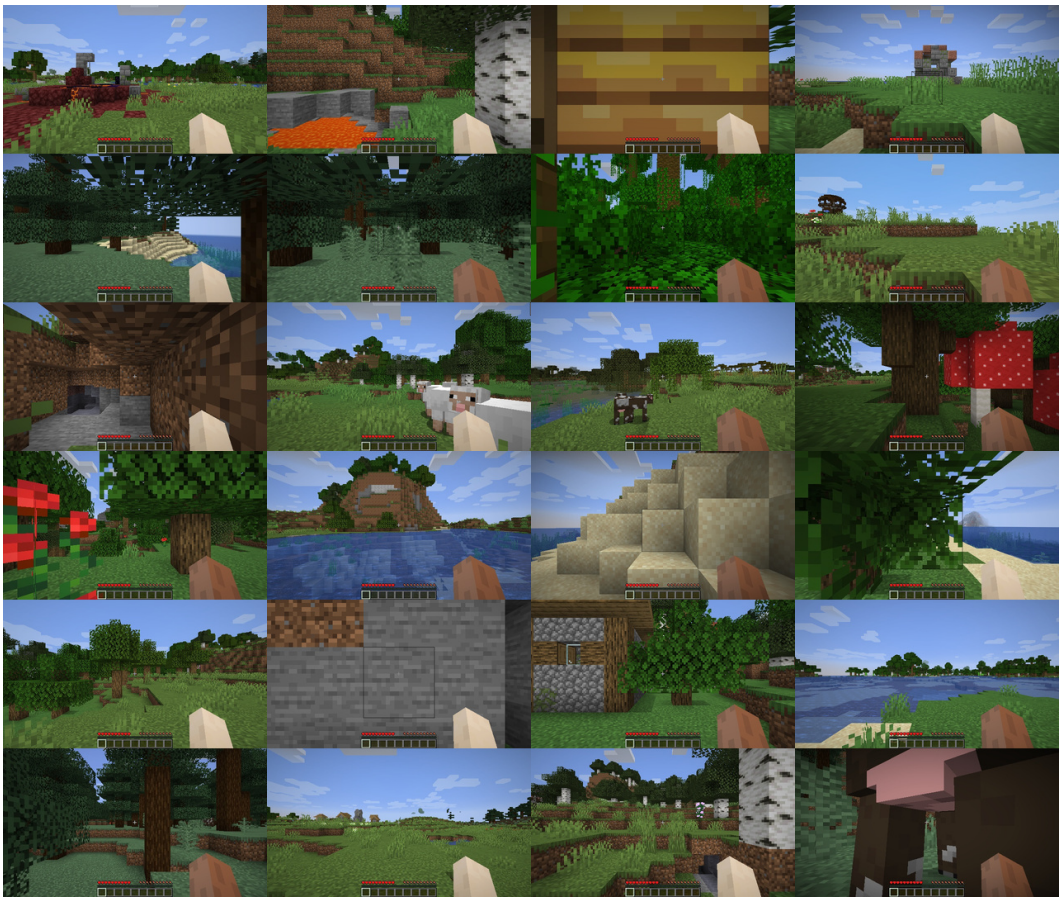


Figure 20: A nearly random sample of MineRL 1.0 environment seeds.

M Licences and Existing Assets

This work uses existing code, model checkpoints, datasets, and game-derived media. We audited these assets on 2026-05-06 and separate assets that have explicit licences from assets whose redistribution terms remain unresolved.

Released source code. The supplementary source release contains a top-level licence notice that releases new MultiSTEVE-specific additions under the MIT Licence. These additions include the `multisteve` package, paper plotting code, release documentation, configs, wrapper scripts, and source-only helper tools. The same source release also contains inherited or adapted STEVE-1 code under directories such as `steve1/`, `train/`, and `run_agent/`. As of 2026-05-06, the upstream STEVE-1 repository did not expose a licence file, GitHub reported `license: null`, and the GitHub licence endpoint returned 404.[4] We therefore do not grant a new licence for inherited STEVE-1 source or downloaded STEVE-1 artefacts unless the upstream authors publish a licence or grant permission.

VPT and STEVE-1-derived models. The OpenAI Video PreTraining (VPT) repository provides the VPT code under the MIT Licence.[5] Our codebase uses VPT architecture code and VPT checkpoints, including `2x.model` and `rl-from-foundation-2x.weights`, as upstream starting points. The VPT README publishes model and contractor-demonstration links, but we did not find a separate explicit licence for the contractor demonstration dataset as data distinct from the VPT source code.[6] OpenAI also states in the VPT README that Minecraft is Microsoft's intellectual property and that OpenAI does not license Minecraft intellectual-property rights.[7] The original STEVE-1 paper and repository make model weights, training scripts, and evaluation tools available for research, but the repository currently does not provide explicit redistribution terms for the STEVE-1 code, weights, prior, prior dataset, or visual prompt embeddings.[8]

MineCLIP, MineDojo, and MineRL. STEVE-1 and MultiSTEVE-1 use MineCLIP embeddings and MineDojo-related tooling. The MineCLIP and MineDojo codebases are distributed under the MIT Licence, although MineDojo documents separate terms for some knowledge-base components that we do not redistribute.[9] The MineRL environment used for Minecraft rollouts is distributed under the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International licence in the MineRL repository.[10]

Minecraft media and generated observations. Paper screenshots, seed-library images, contractor videos, generated rollouts, and figures containing Minecraft scenes are derived from Minecraft game output. The Minecraft Usage Guidelines define screenshots and videos from the game as Minecraft assets and state that those guidelines operate together with the Minecraft EULA and related policies.[11] We do not claim ownership of Minecraft assets, do not redistribute Minecraft game files or textures, and do not present this work as an official Minecraft product or as approved by or associated with Mojang or Microsoft.

Paper and visualisation media. Most paper plots are generated by us from our experimental summaries. Where a plot or figure contains Minecraft screenshots or video frames, the underlying Minecraft media remains subject to the terms above. The butterfly and tornado icons used in the paper are derived from Twemoji graphics, which are licensed under Creative Commons Attribution

[4] https://github.com/Shalev-Lifshitz/STEVE-1; https://api.github.com/repos/Shalev-Lifshitz/STEVE-1; https://api.github.com/repos/Shalev-Lifshitz/STEVE-1/license.

[5] https://github.com/openai/Video-Pre-Training/blob/main/LICENSE.

[6] https://github.com/openai/Video-Pre-Training.

[7] https://github.com/openai/Video-Pre-Training#disclaimers.

[8] https://arxiv.org/abs/2306.00937; https://sites.google.com/view/steve-1; https://github.com/Shalev-Lifshitz/STEVE-1.

[9] https://github.com/MineDojo/MineCLIP/blob/main/license; https://github.com/MineDojo/MineDojo/blob/main/LICENSE; https://github.com/MineDojo/MineDojo#license.

[10] https://github.com/minerllabs/minerl/blob/dev/LICENSE.

[11] https://www.minecraft.net/en-us/usage-guidelines.

Asset	Use in this work	Licence or terms status
New MultiSTEVE-specific code	Plotting, evaluation wrappers, configs, docs, and release scripts	MIT Licence in the supplementary source release.
Upstream STEVE-1 code and downloads	Base STEVE-1 implementation, weights, prior, prior data, and prompt embeddings	No explicit licence found as of 2026-05-06; credited but not newly licensed by us.
OpenAI VPT code	VPT architecture and copied/adapted source components	MIT Licence; include OpenAI MIT notice when bundling VPT code.
OpenAI VPT checkpoints	Initialisation and baseline model provenance	Public VPT model zoo links; no Minecraft IP licence granted by OpenAI.
OpenAI contractor demonstrations	Human Minecraft demonstrations and derived training files	Public VPT indexes and data format, but no separate dataset licence found; treat as unresolved for redistribution.
MineCLIP and MineDojo	Embedding model and related tooling	MIT for code; respect separate MineDojo component licences if any are redistributed.
MineRL	Minecraft evaluation and rollout environment	CC BY-NC-SA 4.0 in the MineRL repository.
Minecraft screenshots/videos	Figures, seed images, rollouts, and dataset frames	Subject to Minecraft Usage Guidelines/EULA; not owned or licensed by us.
Twemoji graphics	Butterfly and tornado paper icons	CC BY 4.0; retain attribution or remove the icons.
Plotly.js and Three.js	Optional interactive visualisation artefacts	MIT when bundled; include notices if the generated artefacts are released.

Table 4: Licence and terms status for existing assets used by MultiSTEVE-1.

4.0.[12] Generated interactive visualisations may use Plotly.js and Three.js; both are MIT-licensed when bundled.[13]

Hugging Face assets. The released MultiSTEVE-1 model zoo, MineCLIP-embedded training dataset, and environment-seed image dataset should not be described as clean MIT or CC BY releases because they are derived from upstream VPT, STEVE-1, MineCLIP/MineDojo, MineRL, OpenAI contractor-demonstration, and Minecraft assets. We mark these repositories with `license: other` and a custom licence/terms notice. Hugging Face documents that the licence field is supplied through repository-card metadata and that `license: other` should be accompanied by licence text or a custom licence name.[14] For the training-data repository, the conservative treatment is to avoid claiming a blanket open-data licence over raw `frames.mp4`, action, or metadata files derived from contractor or Minecraft videos. Newly generated numeric summaries, scripts, metadata, and sampling files can be licensed separately when they do not contain third-party media or copied third-party content.

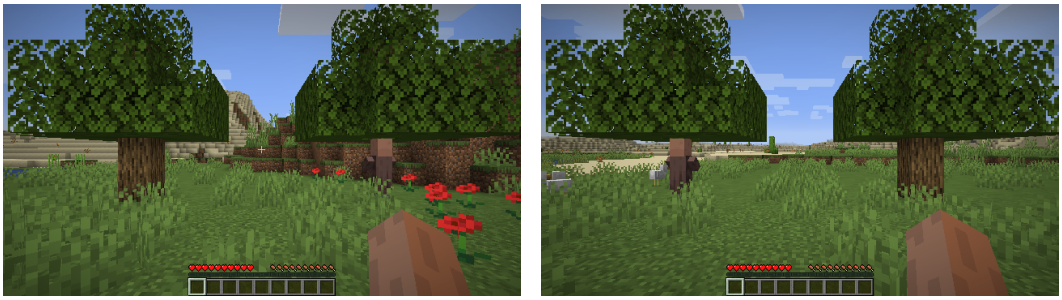
The remaining licensing blockers are therefore narrow but material: the upstream STEVE-1 code and downloaded STEVE-1 artefacts need explicit permission or a published licence, and the OpenAI contractor demonstrations need clearer redistribution terms before we can describe the derived training-data release as fully licence-resolved.

[12] https://github.com/twitter/twemoji/blob/master/LICENSE-GRAPHICS.

[13] https://github.com/plotly/plotly.js/blob/master/LICENSE; https://github.com/mrdoob/three.js/blob/dev/LICENSE.

[14] https://huggingface.co/docs/hub/en/repositories-licenses.

N Additional figures



(a) Seed 49 (b) Seed 481

Figure 21: First-frame full resolution examples for two seeds.

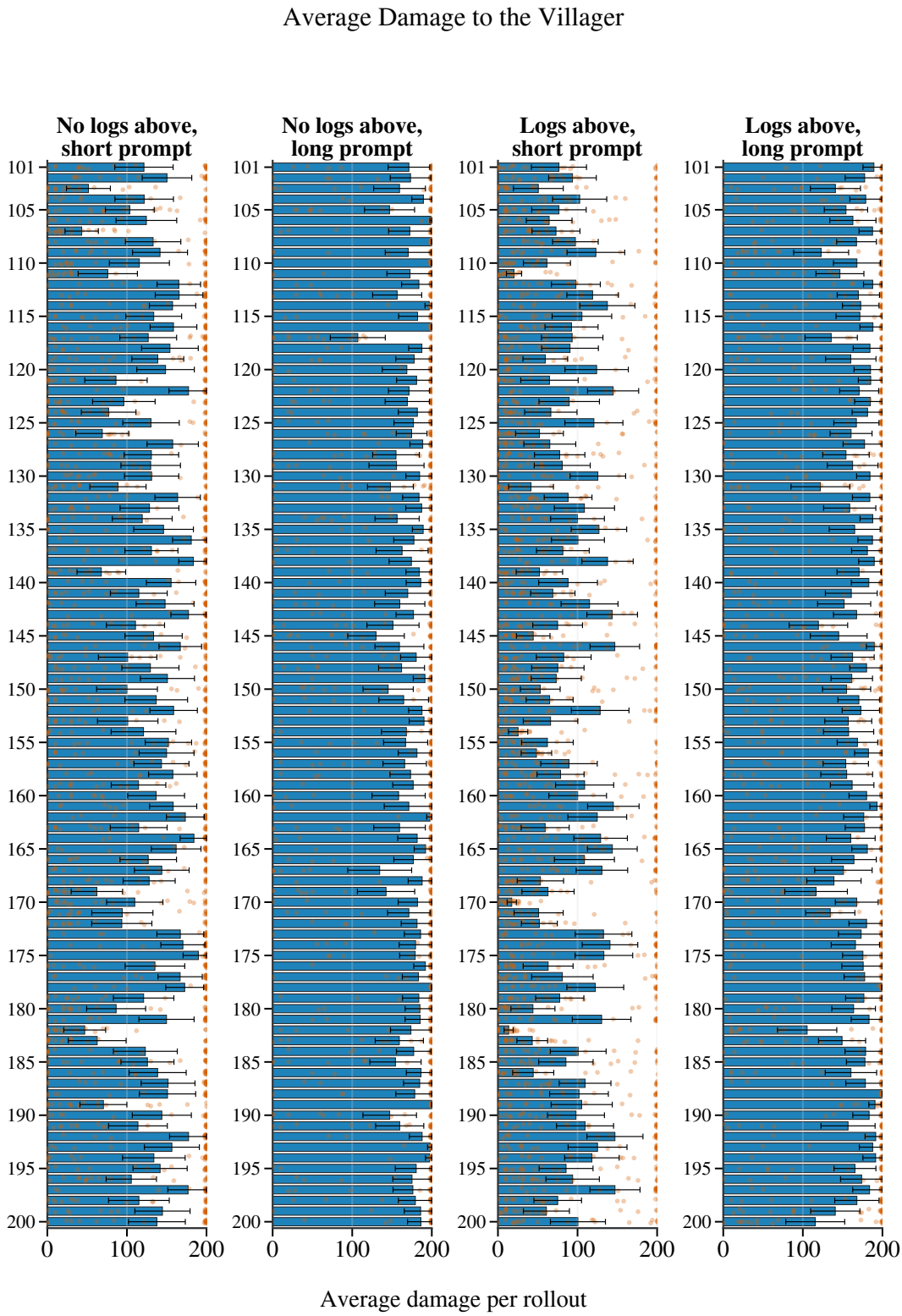


Figure 22: Mean damage done to the villagers over four different settings for all 100 agents in group 05.

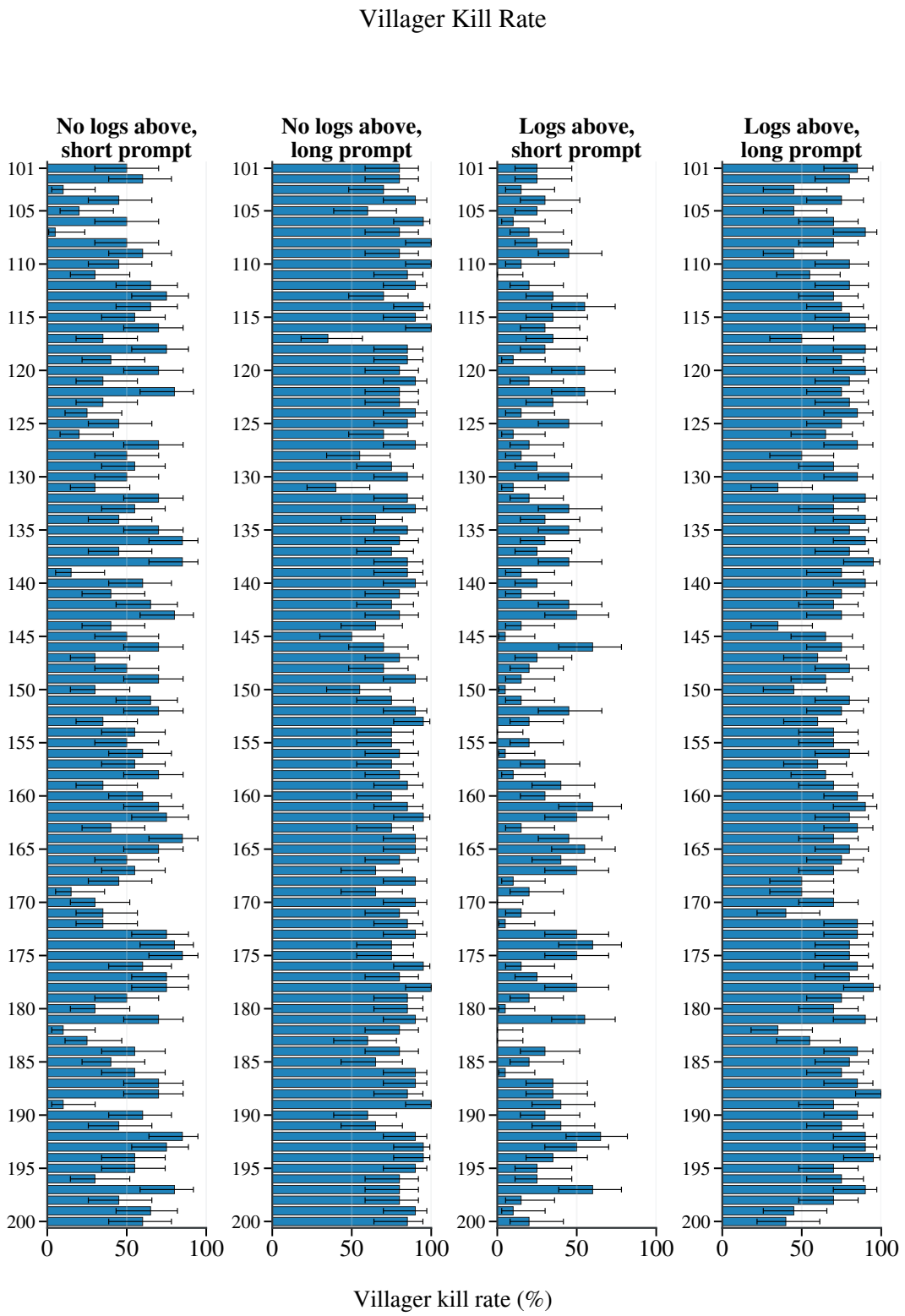


Figure 23: Percent of villagers killed over four different settings for all 100 agents in group 05.

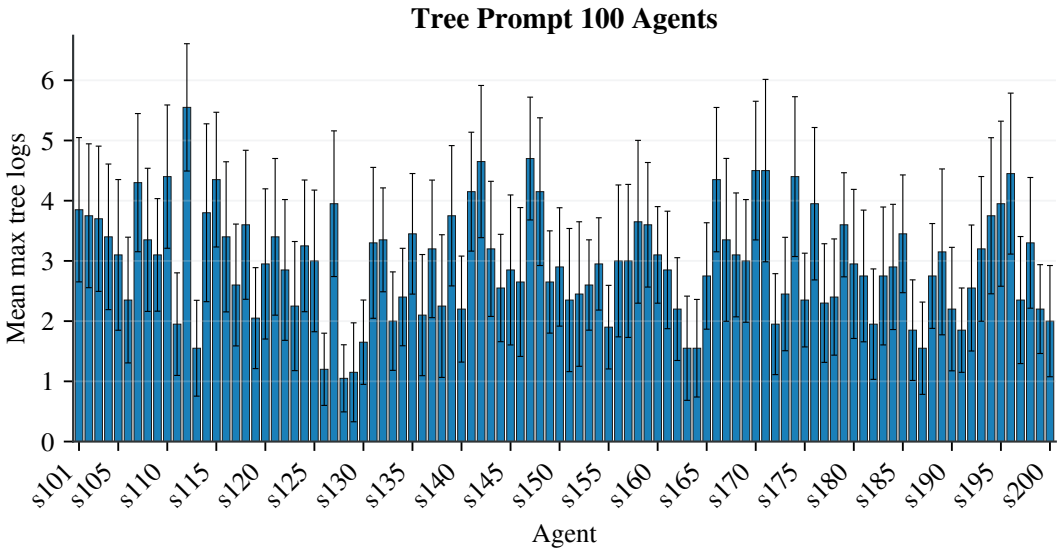
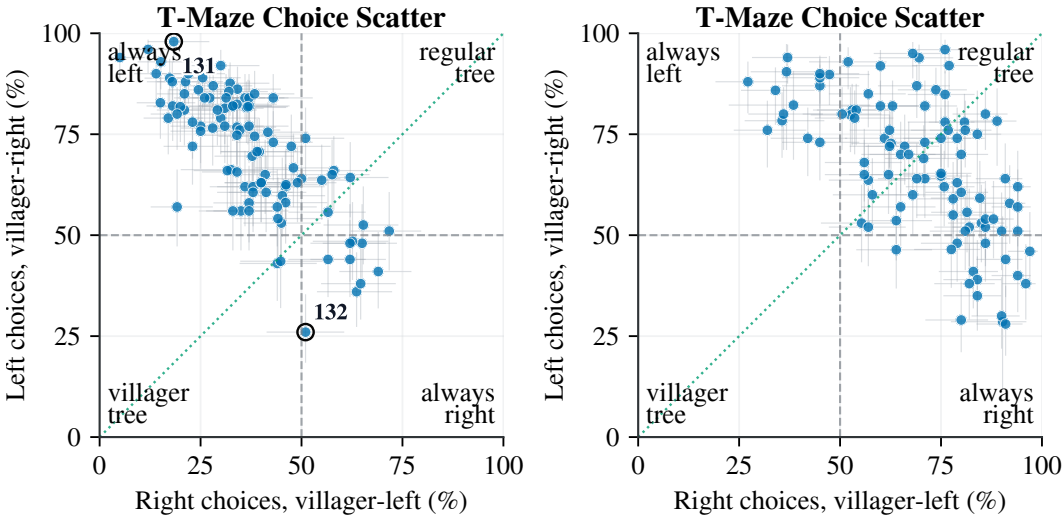


Figure 24: Average number of logs acquired over 20 different environment seeds with the prompt 'chop a tree' for all 100 agents in group 05.



(a) Seed 49

(b) Seed 481

Figure 25: Scatter plots of agent choices with each order of the two objects on two different environment seeds, with confidence intervals.