A SYSTEMATIC STUDY OF BEHAVIORAL CLONING FOR SCIENTIFIC DATA ANNOTATION

Ishaan Singh Chandok*
Department of Physics
Harvard University
Cambridge, MA, USA
ichandok@g.harvard.edu

Core Francisco Park*
Center for Brain Science, Harvard University
Prior Computers
Cambridge, MA, USA
core@priorcomputers.ai

ABSTRACT

Scientific data annotation, such as tracking animals in video or proofreading neural reconstructions, remains bottlenecked by the "last mile" problem: even with strong automation, verification and correction consume substantial human effort. Standard approaches train models to directly predict annotations, discarding the rich supervision in how experts navigate, click, verify, and correct. We introduce a framework for studying behavioral cloning on scientific annotation: 9 synthetic tasks paired with synthetic annotations that simulate realistic human strategies including exploration, mistake correction, and strategic decision-making. Our experiments reveal several findings. First, skills emerge hierarchically: models learn GUI mechanics before task-critical decisions, and commit fewer mistakes than the training data while retaining the ability to correct errors when they occur. Second, scaling models on multi-task behavioral cloning shows that larger models are more data efficient, but exhibit worse decision-making despite similar placement accuracy. Third, multi-task pretraining enables efficient fine-tuning to new tasks, while training from scratch fails entirely. Fourth, linear probes reveal that models internally represent latent variables of the annotation process such as task phase and data position; interestingly, we find a shared mistake representation that generalizes across different annotation tasks. Overall, our framework establishes systematic benchmarks and identifies key bottlenecks, providing a foundation for scaling behavioral cloning to real-world scientific data annotation.

1 INTRODUCTION

Scientific data analysis is bottlenecked by manual annotation, from tracking animals in behavioral videos (Mathis et al., 2018; Pereira et al., 2022; Lauer et al., 2022) to reconstructing wiring diagrams of brains (Scheffer et al., 2020; Consortium et al., 2025). Deep learning has reduced the bulk of annotation, but the "last mile" remains expensive: even when automated systems are mostly correct, humans must still find and fix the remaining errors. In connectomics, for example, proofreading a single fly brain required 33 person-years of effort despite state-of-the-art automated segmentation (Dorkenwald et al., 2022).

Most ML approaches to automate annotation treat the process as a direct mapping from data to labels, optimizing only for final output accuracy. However, human annotation is an interactive workflow: an expert navigates through data, clicks or edits elements in the interface, verifies uncertain cases, and revises their mistakes. These action sequences provide rich supervision about the *process* by which high-quality annotations are produced, not just the final labels. Behavioral cloning (BC) (Pomerleau, 1988; Bain & Sammut, 1995), where models learn to imitate human actions, offers a natural way to exploit this supervision by training directly on annotation traces, an approach that has proven effective for autonomous driving (Bojarski et al., 2016) and game-playing agents (Baker et al., 2022).

Despite BC being a standard approach in robotics and embodied AI (Mandlekar et al., 2021; Chi et al., 2025), its potential for scientific annotation remains poorly characterized. The barriers are largely

*Equal contribution.

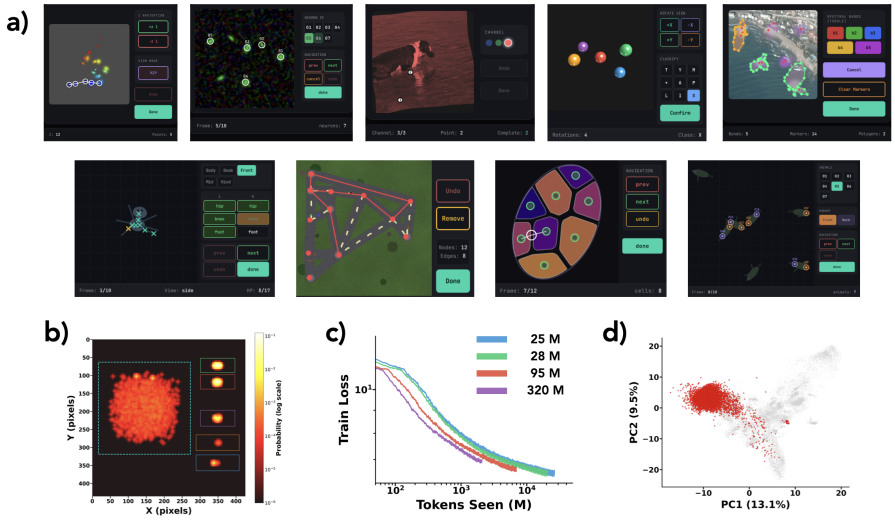


Figure 1: **Framework Overview.** (a) The 9 synthetic annotation tasks: colored dot tracking, neuron tracking, multichannel image alignment, 3D exploration classification, spectral plume finding, animal limb tracking, road network construction, cell lineage tracking, and animal behavior tracking. (b) Click heatmap from a trained model. (c) Scaling behavior: training loss vs. tokens seen for multiple model sizes. Larger models are more data-efficient learners. (d) Internal representations across tasks. Each dot is an activation from a different task; mistakes (red) cluster together regardless of task, indicating a shared mistake representation learned across the multi-task model.

practical: collecting behavioral data requires instrumented interfaces, no standardized benchmarks exist for long-horizon annotation sessions, and uncertain expectations discourage scientists from diverting effort from their core research. As a result, neither ML researchers nor domain scientists have the tools to answer basic questions: How hard is it for a model to learn an annotation workflow? What challenges remain even with substantial data? What factors make some tasks harder than others?

This paper provides a framework for answering these questions. We construct 9 synthetic annotation tasks with procedurally generated data and simulated human behavior, enabling controlled experiments that would otherwise require years of data collection.

Our contributions are:

1. **A multi-task procedural framework for studying behavioral cloning on scientific annotation** (§3, Figure 1). We introduce 9 synthetic tasks inspired by scientific data annotation, paired with virtual annotators that simulate realistic human strategies including information gathering, exploration, mistake-and-correction sequences, and strategic decision-making. A GUI simulator enables both data generation and closed-loop evaluation.
2. **A detailed analysis of training dynamics and skill emergence in single-task learning** (§4.1, Figure 2). We find hierarchical emergence: GUI mechanics (clicking buttons, placing markers) are learned before task-critical decisions (when to undo, when to finalize). Notably, models commit fewer mistakes than present in the training data, yet when teacher-forced into error states, they still correctly trigger undo actions, a beneficial effect of majority-class learning.
3. **Scaling laws and transfer properties for multi-task annotation models** (§4.2–4.3, Figures 3, 4). Larger models are more data-efficient learners: $10\times$ more parameters yields roughly $3\times$ better sample efficiency. However, larger models show similar placement accuracy but *worse* decision-making from visual features. Multi-task pretrained models transfer efficiently via fine-tuning, while in-context learning and training from scratch both fail entirely.
4. **Interpretability analysis revealing shared mistake representations across tasks** (§4.4, Figure 5). We find that mistake representations show cross-task transfer: a pooled probe achieves 0.87 ROC AUC, and probes trained on 8 tasks transfer to held-out tasks with mean 0.71 ROC AUC.

One exception is 3D exploration (0.29), where errors are classification mistakes rather than spatial placement errors—suggesting the shared representation is specific to placement-type errors rather than a fully task-general "mistake" concept. Models also encode annotation phase, data-relative time (position within the data, not sequence position), and task state.

2 Related Work

Behavioral Cloning and GUI Agents. Behavioral cloning (Pomerleau, 1988; Bain & Sammut, 1995) learns policies via supervised learning on expert demonstrations. Classic challenges include covariate shift, addressed by DAgger (Ross et al., 2011), and multimodal action distributions, handled by diffusion policies (Chi et al., 2025). Sequence modeling approaches like Decision Transformer (Chen et al., 2021) and Trajectory Transformer (Janner et al., 2021) frame action prediction as next-token prediction, while multi-task agents like Gato (Reed et al., 2022) extend this to diverse domains. Deep learning revived interest in end-to-end BC for driving (Bojarski et al., 2016) and manipulation (Mandlekar et al., 2021), with vision-language-action models like RT-2 (Brohan et al., 2023) and OpenVLA (Kim et al., 2024) transferring web-scale knowledge to robotic control. In parallel, GUI agents learn to navigate computer interfaces through clicks and keystrokes; systems like WebArena (Zhou et al., 2024) and SeeClick (Cheng et al., 2024) train models to complete web tasks from screenshots. Our setting resembles GUI agents but focuses on scientific annotation rather than general web navigation, providing cleaner evaluation (ground truth is known) and exposing distinct challenges like long-horizon credit assignment and rare critical actions.

Scientific Annotation Automation. Automation in scientific imaging has primarily pursued direct prediction: flood-filling networks for connectomics (Januszewski et al., 2018), tracking algorithms for cells and animals (Lauer et al., 2022), pose estimators for keypoints (Mathis et al., 2018; Pereira et al., 2022). These approaches discard the rich supervision in how humans actually annotate: navigation patterns, verification habits, and error correction strategies. Interactive methods like SAM (Kirillov et al., 2023) incorporate user clicks but model single refinement interactions rather than full annotation sessions. Active learning (Settles, 2009) reduces annotation burden by selecting informative samples but still treats annotation as atomic labeling events.

Synthetic Data and Positioning. Synthetic data enables controlled study of learning dynamics (Chan et al., 2022; Raventós et al., 2023; Allen-Zhu & Li, 2023). We use synthetic annotation data to isolate factors confounded in real studies: task difficulty, behavior patterns, error rates. The closest methodological analog is Video PreTraining (VPT) (Baker et al., 2022), which applies behavioral cloning to Minecraft gameplay. We adapt this approach to scientific annotation, using synthetic data to overcome the absence of large-scale behavioral datasets. Prior work has rarely focused on training from full annotation sessions, the sequences of navigation, placement, verification, and correction that constitute expert behavior.

3 Framework

Training behavioral cloning models requires interleaved sequences of GUI screenshots and user actions: $(\text{img}_0, \text{click}_0, \text{img}_1, \text{click}_1, \ldots)$. We generate this data by separating three concerns: *what exists in the world* (task instances with ground truth), *how a human would annotate it* (action sequences), and *how to render it* (GUI screenshots). This separation enables controlled experiments: we can vary task difficulty, annotation behavior, and error rates independently.

Virtual Human Annotation Model. The key component is a procedural model of human annotation behavior. Given a task instance with ground truth, it generates realistic action sequences including:

- **Navigation**: moving through z-slices, rotating 3D views, switching spectral channels
- **Placement**: clicking to place markers, draw polygons, or select objects
- **Verification**: checking work by toggling views or revisiting previous locations
- **Mistakes and corrections**: simulated errors (10–30% of placements, varying by task) followed by undo actions

This captures supervision about the annotation *process*, the navigation patterns, verification habits, and error recovery strategies that constitute expert behavior, rather than just the final output.

Task Suite. We construct 9 tasks spanning diverse annotation challenges. *Tracking tasks* require maintaining object identity across frames: colored dots through z-slices (connectomics-inspired), neurons through elastic deformation, cells through division events, and animals through independent motion. *Animal limb tracking* requires inferring keypoint positions from rendered morphology without explicit markers. *Multichannel alignment* tests cross-channel correspondence under geometric distortion. *3D exploration* requires active information gathering, rotating an occluded object before classification. *Road network construction* involves graph annotation (nodes at intersections, edges along roads). *Spectral plume finding* combines multi-band reasoning with polygon drawing. Full task details appear in Appendix A.

Model Architecture. We use a vision-language model (Alayrac et al., 2022; Liu et al., 2023) that processes interleaved screenshots and click coordinates. A DINOv2 vision encoder (Oquab et al., 2024) extracts patch tokens from each frame, spatially pooled to $12 \times 9 = 108$ tokens. A transformer head processes the sequence with *block-causal attention*: bidirectional within each frame's patches, but causal across frames. The model predicts the next click as independent distributions over x and y coordinates. We train four model sizes (25M to 320M parameters) to study scaling. Notably, off-the-shelf vision-language models cannot handle annotation-length contexts; even Qwen3-VL-4B-Instruct (Bai et al., 2025) exceeds 80GB GPU memory at 20 frames. Our architecture retains spatial structure (required for pixel-precise prediction) while remaining trainable. Full details appear in Appendix B.

Evaluation. We evaluate in two modes. *Teacher-forced evaluation* measures next-action prediction accuracy given ground-truth history, enabling fine-grained analysis of per-action-type learning without compounding errors. *Autoregressive evaluation* runs the model in closed loop with a GUI simulator: the model predicts clicks from screenshots, the simulator executes them through the GUI's JavaScript interface, and the process repeats until task completion or timeout. Critically, the simulator determines action type from coordinates alone; the model cannot specify action types directly, ensuring evaluation measures whether it has learned *where* to click.

4 RESULTS

4.1 SINGLE TASK MODEL

We first analyze how models learn annotation behavior on a single task (colored dot tracking, depicted in Figure 2a) to understand the dynamics of skill acquisition. Unlike in natural language modeling, GUI annotation tasks decompose into clearly identifiable sub-skills: clicking specific buttons, navigating a dataset, placing markers at precise locations, and correcting errors. Training a base model (95M parameters) on colored dot tracking provides insight into how these skills are acquired.

Figure 2b (left) shows training dynamics: total loss decomposes into x and y coordinate losses, while canvas click accuracy is measured at both coarse (5px) and fine (1px) precision. Fine placement accuracy emerges sharply only after sufficient loss reduction, suggesting a phase transition in spatial precision.

Skills emerge in a hierarchy. Figure 2b (right) depicts teacher-forced accuracy per action type across training. General GUI understanding and button usage emerge quickly, as the model quickly learns to click valid portions of the interface and use navigation buttons ($+$z, $-$z). In contrast, capabilities requiring complex visual reasoning emerge more slowly: placement accuracy improves gradually, and undo accuracy (which requires recognizing that a mistake was made) emerges later still. Notably, undo accuracy eventually reaches near 100%, indicating the model learns to perfectly identify and correct errors when teacher-forced into mistake states.

We also studied the order in which skills are learned via generative evaluation (Figure 2c, left). GUI validity is consistently high throughout training. Navigation emerges quickly, as the model quickly learns to use $+$z to find and click the first blue dot. Dot placement accuracy improves more gradually

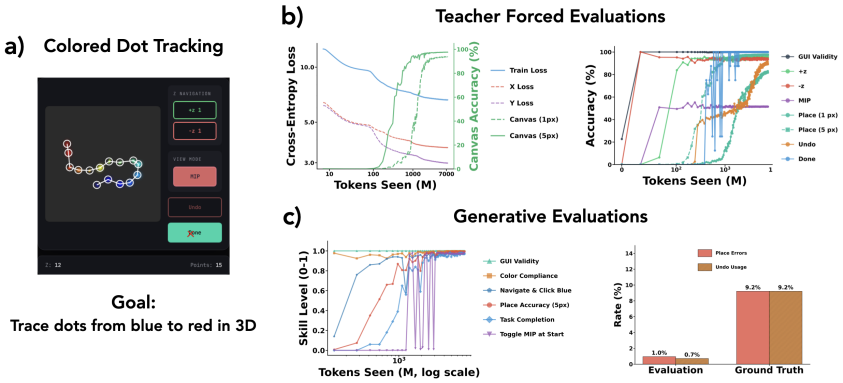


Figure 2: **Single Task Model Analysis.** (a) The colored dot tracking task: dots are scattered in 3D from blue (start) to red (end); the annotator uses $+$z/$-$z buttons to navigate in depth and clicks dots in color order, then clicks Done. (b) Left: training loss (total, x, y) and canvas click accuracy at 1px and 5px precision. Right: teacher-forced evaluation shows when capabilities emerge. (c) Left: generative evaluation of skill acquisition; GUI validity is always high, navigation emerges fast, dot placement improves gradually, but MIP button usage (showing maximum intensity projection) emerges very late despite having a sharp transition. Right: models naturally make fewer mistakes than the training data distribution, a known phenomenon where models under-represent minorities; yet teacher-forced evaluation (b) confirms the model can still correct errors when they occur.

through training. Interestingly, MIP button usage (which toggles a maximum intensity projection view) emerges very late despite exhibiting a sharp transition when it does appear. This stereotyped behavior from the training data (toggling MIP at episode start) requires the model to learn a specific sequential pattern rather than respond to immediate visual cues, explaining its delayed acquisition.

Some skills exhibit non-monotonic learning. Done accuracy in teacher-forced evaluations (Figure 2b (right)) and the model's ability to toggle MIP at the start of generative evaluation (Figure 2c (left)) oscillate through learn-unlearn cycles before stabilizing. This pattern is consistent with newly acquired skills temporarily interfering with existing ones until the model finds a stable configuration. See Figure 18 for individual plots with error bars.

Models under-represent mistakes, but still know how to correct them. Figure 2c (right) reveals a striking pattern: models naturally make far fewer mistakes than present in the training data. Training data includes mistakes followed by undo clicks (9.2% of place actions), but trained models in generative evaluation make errors only 1.0% of the time and rarely use undo (0.7%). This is a known phenomenon where models amplify dataset biases by under-representing minority classes (Zhao et al., 2017), effectively learning to "skip" the mistake-then-correct pattern. One might conclude the model simply cannot handle errors. However, teacher-forced evaluation (Figure 2b, right) shows otherwise: when forced into a mistake state, the model predicts undo with near-perfect accuracy.

4.2 Multi Task Model

We trained four model sizes (25M, 28M, 95M, 320M parameters) on all 9 tasks jointly. Larger models are more data-efficient, achieving lower loss with fewer tokens seen (Figure 3a), consistent with language model scaling laws. However, at our compute budget, we did not reach the regime where larger models become compute-efficient; the 25M model trained longer outperforms the 320M model at equal FLOPs (Figure 21).

Comparing models at equal training loss reveals a surprising result: smaller models are *better* at decision actions (Figure 3b, left). When choosing whether to click done, undo a mistake, or navigate between frames, smaller models consistently outperform larger ones. In contrast, motor actions (placing markers at correct pixel locations, using navigation utilities) show no model size dependence (Figure 3b, right). We hypothesize that smaller models, forced to compress information due to limited

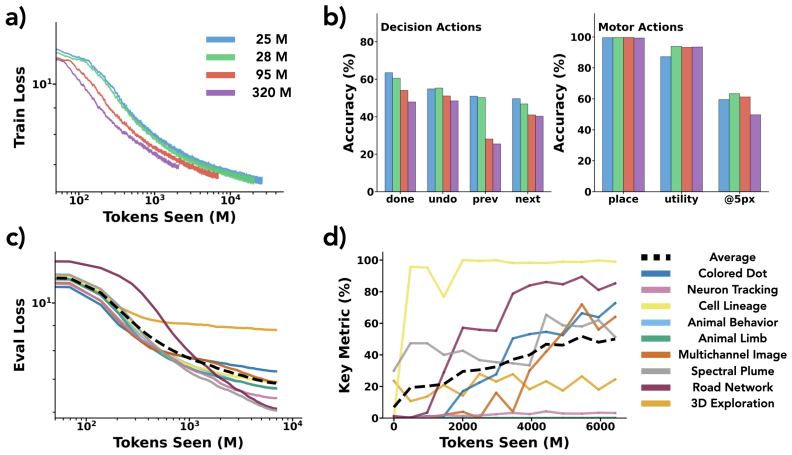


Figure 3: **Multi-task Model Analysis.** (a) Training loss vs. tokens seen for four model sizes (25M–320M parameters). Larger models are more data-efficient, achieving lower loss with fewer tokens. (b) Left: Decision action accuracy (done, undo, prev, next) at equal loss: surprisingly, smaller models outperform larger models. Right: Motor action accuracy (placing markers, using navigation) shows no model size dependence. (c) Per-task loss decomposition during training. Different tasks are learned at different times; 3D exploration dominates late-training loss despite containing learnable features (Section 4.4). (d) Per-task evaluation metrics during training (see Appendix for metric definitions). The smooth aggregate improvement masks discrete task-level breakthroughs occurring at different times.

capacity, learn the abstract decision-making factors that transfer across contexts, while larger models can afford to memorize surface patterns.

Decomposing loss by task during training (Figure 3c) reveals that different tasks are learned at different times, reminiscent of the quanta hypothesis in neural scaling (Michaud et al., 2024; Kangaslahti et al., 2025). Early in training, simpler tasks like neuron tracking improve rapidly. Later, 3D exploration classification dominates the loss and, notably, stops improving despite not being solved. In Section 4.4, we show that the model *does* learn the correct classification features internally, yet fails to use them for prediction, consistent with recent findings that VLMs can encode visual information they fail to act upon (Fu et al., 2025).

Per-task evaluation metrics (Figure 3d; see Appendix for metric definitions) show that the smooth aggregate improvement curve masks discrete task-level breakthroughs occurring at different training stages. This staged acquisition suggests that multi-task behavioral cloning, like language modeling, learns skills in a structured order determined by task complexity and data statistics.

4.3 Downstream Adaptation

To test whether multi-task pretraining produces models that adapt efficiently to new tasks, we evaluate on a held-out shape matching task not seen during training (Figure 4a). This task requires clicking all objects matching a template shown in a sidebar. Unlike all training tasks, which have button elements on the right side of the GUI, this task places them on the left.

Fine-tuning the pretrained multi-task model on 500 sequences from the new task achieves strong performance (76.6% accuracy, Figure 4b). In contrast, training the same architecture from scratch on 7800 sequences, over 15 times more data, achieves 0% accuracy. This gap suggests synthetic pretraining followed by fine-tuning on limited real data as a practical path forward.

However, in-context learning fails entirely (Figure 4b). We tested whether the pretrained model could adapt through examples provided in context, either as demonstration sequences or as prefixes of

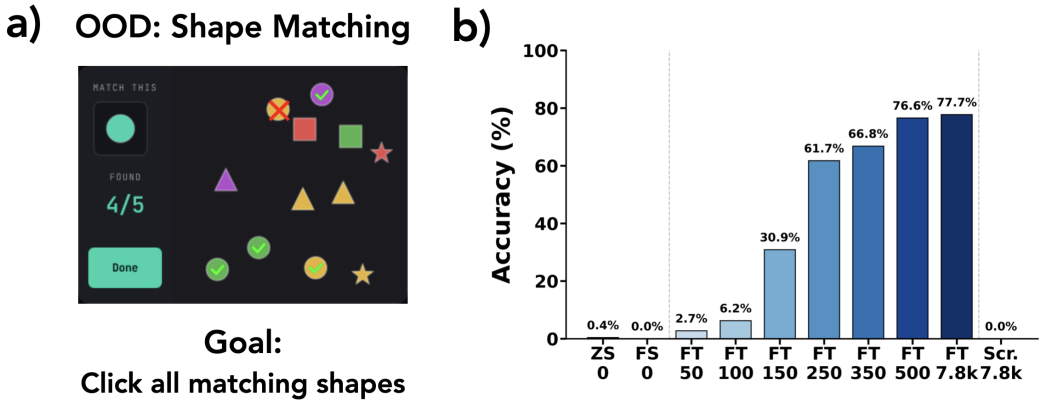


Figure 4: **Downstream Adaptation.** (a) The held-out shape matching task: click all objects matching the template in the sidebar. This task was not seen during training and has a different GUI layout. (b) Evaluation across adaptation methods. Zero-shot (ZS) and few-shot (FS) in-context learning achieve negligible accuracy. Fine-tuning saturates at 500 sequences (76.6%); 7,800 sequences yields similar performance. Training from scratch fails entirely (0%) even with 7,800 sequences.

ground-truth actions. In all cases, accuracy remained at 0%. The model can transfer via fine-tuning but cannot learn from in-context examples, possibly due to limited task diversity (9 tasks).

4.4 Model Internal Analysis

Having trained models on annotation behavior, we examine what representations they learn using linear probing (Li et al., 2022; Gurnee & Tegmark, 2023; Nanda et al., 2023).

We first examine whether models represent mistakes and corrections. We train linear probes on the residual stream activations at the [cls] token (the click-generating position) to predict whether the upcoming action is a mistake or a correction. Despite the severe class imbalance ($<$5% of actions are mistakes), we use ROC AUC rather than accuracy as our metric, since a classifier that simply predicts "no mistake" for all inputs would achieve $>$95% accuracy but 0.50 ROC AUC. We find that both mistake and correction states are linearly separable: the mistake probe achieves 0.92 ROC AUC on single-task data and 0.87 when pooled across all 9 tasks; the correction probe achieves 0.99 ROC AUC. As shown in Figure 5a, projecting activations onto the learned mistake and correction directions reveals clear clustering: mistakes (red) separate along the mistake axis while corrections (blue) separate along the correction axis, with correct actions (gray) distributed throughout.

Beyond mistakes, we probe for other latent variables relevant to annotation (Figure 5b). These include: annotation phase in road network construction (node placement vs. edge construction; ROC AUC = 1.00), object class in 3D exploration (9-way classification; ROC AUC = 0.93), color progress in colored dot tracking ($R^2 = 0.92$), frame position in neuron and behavioral tracking (ROC AUC = 0.70–0.89), and marker progress in image alignment. Importantly, variables like "frame position" are not simply the number of frames observed; annotators navigate back and forth with prev/next buttons, so the model must track position within the data rather than just count inputs. Similarly, color progress reflects semantic progress through the annotation task, not merely temporal sequence position. All probes perform well above chance, indicating that models develop internal representations of task-relevant state.

A natural question is whether these representations are task-specific or shared across tasks. Mistake detection, for instance, could in principle be computed from visual features alone (e.g., a marker placed on empty space). To test for shared structure, we pool activations from all 9 tasks and visualize via PCA (Figure 5c). Strikingly, mistakes cluster together across different tasks, suggesting a partially universal "something is wrong" representation that transcends task-specific error patterns.

To quantify cross-task generalization, we train mistake probes on 8 tasks and evaluate on the held-out task (Figure 5d). Mean transfer ROC AUC = 0.71, with 8 of 9 tasks showing above-chance transfer. The exception is 3D exploration classification (ROC AUC = 0.29), which is below the 0.5 chance level, indicating the probe has learned a direction that is *anti-correlated* with mistakes in this task. This makes sense: 3D exploration mistakes are classification errors (clicking the wrong class button), fundamentally different from spatial placement errors in other tasks. The model appears to encode these distinct error types in opposing directions.

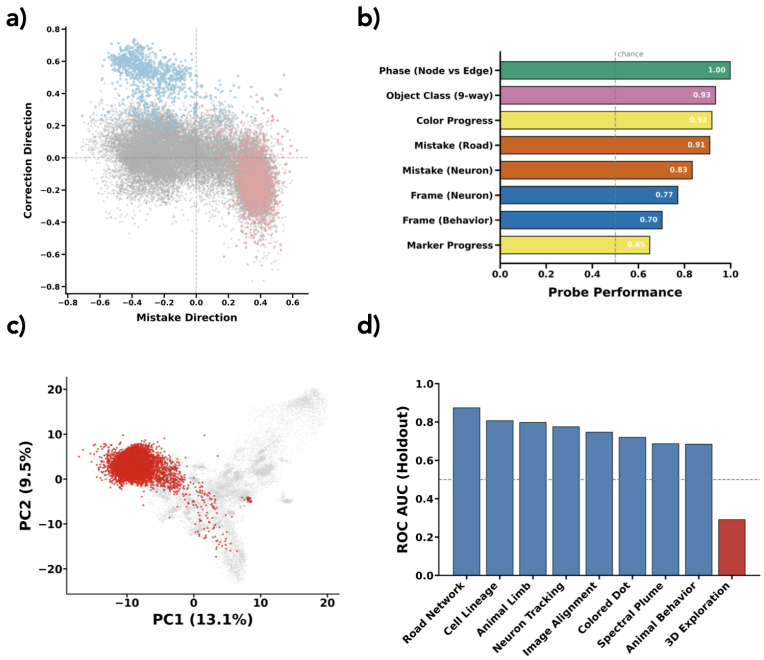


Figure 5: **Model Internals.** (a) Projection of activations onto learned mistake and correction directions. Mistakes (red) cluster with high mistake scores; corrections (blue) cluster with high correction scores; correct actions (gray) are distributed throughout. The orthogonal clustering demonstrates distinct representations for error states versus recovery actions (mistake probe ROC AUC = 0.87, correction probe ROC AUC = 0.99). (b) Linear probes decode diverse information from model activations: task phase (node vs. edge placement, AUC = 1.00), object identity (9-way classification, AUC = 0.93), task progress (R^2 = 0.65–0.92), error states (AUC = 0.83–0.91), and temporal position (AUC = 0.70–0.77). All probes perform well above chance (dashed line). (c) PCA of activations pooled across all 9 tasks. Mistakes (red) form a distinct cluster separate from correct actions (gray), suggesting a partially universal mistake representation. (d) Leave-one-task-out cross-validation: probes trained on 8 tasks and tested on the held-out task. 8/9 tasks show above-chance transfer (blue bars, mean AUC = 0.71), while 3D exploration shows poor transfer (red bar), indicating both generalizable and task-specific components of mistake detection.

5 Discussion

How models learn annotation behavior is central to scaling behavioral cloning for scientific applications. Our analysis reveals that standard maximum likelihood training creates systematic blind spots: rare but critical actions like error correction are underweighted, leading to models that can correct mistakes when prompted but choose not to during generation. Understanding when and how behaviors emerge during training (specifically, the hierarchical acquisition we observe) could guide better data curation strategies. Oversampling critical decision points, or using loss weighting that reflects action importance rather than frequency, may address these bottlenecks.

Our findings also have implications for interface design. Current annotation GUIs are designed for humans alone. But if we understand what makes behaviors hard for models to learn (sparse critical actions, long-horizon dependencies, ambiguous decision points), we can design AI-native interfaces that make these patterns more frequent or more learnable. For instance, an explicit "verify" button might be easier to learn than an implicit toggle-check-toggle pattern. The synthetic framework we provide enables evaluating such designs before deployment.

More broadly, we hope this work helps resolve the collective action problem that has hindered progress on behavioral cloning for scientific annotation. The uncertainty around scaling (how much data is needed, what performance to expect, which tasks are tractable) has discouraged both ML researchers and domain scientists from investing in this direction. By providing systematic baselines across controlled tasks, we offer a foundation for making informed decisions about when behavioral cloning is viable and how to scale it effectively.

Limitations. Our study is synthetic-only; transfer to real annotation is unvalidated. However, our goal is to establish the science before committing to large-scale data collection, and the framework enables informed decisions about when BC is viable. We use pure behavioral cloning without RL; reward shaping may address rare action underweighting. In-context learning failed entirely; inducing ICL for procedural tasks remains open. Our architecture is memory-intensive, limiting context length; efficient alternatives merit exploration.

REFERENCES

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katie Millican, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangooei, Marianne Monteiro, Jacob Menick, Sebastian Borgeaud, Andrew Brock, Aida Nematzadeh, Sahand Sharifzadeh, Mikolaj Binkowski, Ricardo Barreira, Oriol Vinyals, Andrew Zisserman, and Karen Simonyan. Flamingo: a visual language model for few-shot learning, 2022. URL https://arxiv.org/abs/2204.14198.

Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 1, learning hierarchical language structures. *ArXiv e-prints, abs/2305.13673, May*, 2023.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mei Li, Mingsheng Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren, Sibo Song, Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. Qwen3-vl technical report, 2025. URL https://arxiv.org/abs/2511.21631.

Michael Bain and Claude Sammut. A framework for behavioural cloning. In *Machine intelligence 15*, pp. 103–129, 1995.

Bowen Baker, Ilge Akkaya, Peter Zhokov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon Houghton, Raul Sampedro, and Jeff Clune. Video pretraining (vpt): Learning to act by watching unlabeled online videos. *Advances in Neural Information Processing Systems*, 35:24639–24654, 2022.

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

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alexander Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Lisa Lee, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, Henryk Michalewski, Igor Mordatch, Karl Pertsch, Kanishka Rao, Krista Reymann, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Pierre Sermanet, Jaspiar Singh, Anikait Singh, Radu Soricut, Huong Tran, Vincent Vanhoucke, Quan Vuong, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Jialin Wu, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. Rt-2: Vision-language-action models transfer web knowledge to robotic control, 2023. URL https://arxiv.org/abs/2307.15818.

Stephanie C. Y. Chan, Adam Santoro, Andrew K. Lampinen, Jane X. Wang, Aaditya Singh, Pierre H. Richemond, Jay McClelland, and Felix Hill. Data distributional properties drive emergent in-context learning in transformers, 2022. URL https://arxiv.org/abs/2205.05055.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling, 2021. URL https://arxiv.org/abs/2106.01345.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. Seeclick: Harnessing gui grounding for advanced visual gui agents, 2024. URL https://arxiv.org/abs/2401.10935.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.

MICrONS Consortium et al. Functional connectomics spanning multiple areas of mouse visual cortex. *Nature*, 640(8058):435–447, 2025.

Timothée Darcet, Maxime Oquab, Julien Mairal, and Piotr Bojanowski. Vision transformers need registers, 2023. URL https://arxiv.org/abs/2309.16588.

Sven Dorkenwald, Claire E McKellar, Thomas Macrina, Nico Kemnitz, Kisuk Lee, Ran Lu, Jingpeng Wu, Sergiy Popovych, Eric Mitchell, Barak Nehoran, et al. Flywire: online community for whole-brain connectomics. *Nature methods*, 19(1):119–128, 2022.

Stephanie Fu, Tyler Bonnen, Devin Guillory, and Trevor Darrell. Hidden in plain sight: Vlms overlook their visual representations, 2025. URL https://arxiv.org/abs/2506.08008.

Wes Gurnee and Max Tegmark. Language models represent space and time. *arXiv preprint arXiv:2310.02207*, 2023.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem, 2021. URL https://arxiv.org/abs/2106.02039.

Michał Januszewski, Jörgen Kornfeld, Peter H Li, Art Pope, Tim Blakely, Larry Lindsey, Jeremy Maitin-Shepard, Mike Tyka, Winfried Denk, and Viren Jain. High-precision automated reconstruction of neurons with flood-filling networks. *Nature methods*, 15(8):605–610, 2018.

Sara Kangaslahti, Elan Rosenfeld, and Naomi Saphra. Hidden breakthroughs in language model training, 2025. URL https://arxiv.org/abs/2506.15872.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. Openvla: An open-source vision-language-action model, 2024. URL https://arxiv.org/abs/2406.09246.

Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. *arXiv preprint arXiv:2304.02643*, 2023.

Jessy Lauer, Mu Zhou, Shaokai Ye, William Menegas, Steffen Schneider, Tanmay Nath, Mohammed Mostafizur Rahman, Valentina Di Santo, Daniel Soberanes, Guoping Feng, et al. Multi-animal pose estimation, identification and tracking with deeplabcut. *Nature Methods*, 19(4): 496–504, 2022.

Kenneth Li, Aspen K Hopkins, David Bau, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Emergent world representations: Exploring a sequence model trained on a synthetic task. In *The Eleventh International Conference on Learning Representations*, 2022.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning, 2023. URL https://arxiv.org/abs/2304.08485.

Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. *arXiv preprint arXiv:2108.03298*, 2021.

Alexander Mathis, Pranav Mamidanna, Kevin M Cury, Taiga Abe, Venkatesh N Murthy, Mackenzie Weygandt Mathis, and Matthias Bethge. Deeplabcut: markerless pose estimation of user-defined body parts with deep learning. *Nature neuroscience*, 21(9):1281–1289, 2018.

Eric J. Michaud, Ziming Liu, Uzay Girit, and Max Tegmark. The quantization model of neural scaling, 2024. URL https://arxiv.org/abs/2303.13506.

Neel Nanda, Andrew Lee, and Martin Wattenberg. Emergent linear representations in world models of self-supervised sequence models. In *Proceedings of the 6th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pp. 16–30, 2023. URL https://arxiv.org/abs/2309.00941.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. Dinov2: Learning robust visual features without supervision, 2024. URL https://arxiv.org/abs/2304.07193.

Talmo D Pereira, Nathaniel Tabris, Arie Matsliah, David M Turner, Junyu Li, Shruthi Ravindranath, Eleni S Papadoyannis, Edna Normand, David S Deutsch, Z Yan Wang, et al. Sleap: A deep learning system for multi-animal pose tracking. *Nature methods*, 19(4):486–495, 2022.

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

Allan Raventós, Mansheej Paul, Feng Chen, and Surya Ganguli. Pretraining task diversity and the emergence of non-bayesian in-context learning for regression, 2023. URL https://arxiv.org/abs/2306.15063.

Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, Tom Eccles, Jake Bruce, Ali Razavi, Ashley Edwards, Nicolas Heess, Yutian Chen, Raia Hadsell, Oriol Vinyals, Mahyar Bordbar, and Nando de Freitas. A generalist agent, 2022. URL https://arxiv.org/abs/2205.06175.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635. JMLR Workshop and Conference Proceedings, 2011.

Louis K Scheffer, C Shan Xu, Michal Januszewski, Zhiyuan Lu, Shin-ya Takemura, Kenneth J Hayworth, Gary B Huang, Kazunori Shinomiya, Jeremy Maitlin-Shepard, Stuart Berg, et al. A connectome and analysis of the adult drosophila central brain. *elife*, 9:e57443, 2020.

Burr Settles. Active learning literature survey. 2009.

Jieyu Zhao, Tianlu Wang, Mark Yatskar, Vicente Ordonez, and Kai-Wei Chang. Men also like shopping: Reducing gender bias amplification using corpus-level constraints, 2017. URL https://arxiv.org/abs/1707.09457.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents, 2024. URL https://arxiv.org/abs/2307.13854.

Appendix

A TASKS

A.1 FRAMEWORK

Training behavioral cloning models requires interleaved sequences of GUI screenshots and user actions: $(\text{img}_0, \text{click}_0, \text{img}_1, \text{click}_1, \ldots)$. We generate this data by separating three concerns: *what exists in the world* (task instances), *how a human would annotate it* (action sequences), and *how to render it* (GUI screenshots). This section describes the pipeline.

Architecture Overview. Our framework consists of three core components:

1. **Task Instance Generator**: Produces the object to be annotated and its ground truth. This may be purely synthetic (e.g., 3D point clouds, procedural road networks) or derived from real images (e.g., natural images with synthetic deformations).
2. **Virtual Human Annotation Model**: Simulates human annotation behavior given a task instance. Because we have access to ground truth, we can efficiently generate realistic action sequences that mimic human patterns: exploration before committing, verification after placement, occasional mistakes followed by corrections. We can also perturb parameters (error rates, verification frequency, navigation strategies) to study how behavioral variations affect learning.
3. **GUI Renderer**: Takes both task instances and action sequences, then renders the interleaved image-action training data. The GUI is defined in HTML/CSS/JavaScript; a headless browser (Playwright + Chromium) executes the rendering.

The 5-Step Pipeline. Each task is implemented as a 5-step pipeline:

Step	Name	Output
1	Download Data	External datasets (optional)
2	Prepare Instances	Task instance JSONs
3	Make Human Annotations	Action sequence JSONs
4	Make ML Dataset	PNG images + metadata CSV
5	Visualize	Diagnostic figures

Table 1: **Pipeline Steps.** Steps for each task.

Virtual Human Annotation Model. Step 3 implements a procedural model of human annotation behavior. Given a task instance, it generates the complete sequence of actions a human would perform, including:

- **Navigation**: Moving through z-slices, rotating 3D views, switching channels
- **Placement**: Clicking to place markers, draw polygons, or select objects
- **Verification**: Checking work by toggling views or revisiting previous locations
- **Mistakes and corrections**: Simulated errors (10–30% of placements, varying by task) followed by undo actions

The annotation model operates entirely on abstract task data. For example, in the colored dot tracking task, it receives a list of 3D points with colors and decides: "navigate to slice 7, place marker at (0.45, 0.62), toggle MIP view to verify, navigate to slice 9…" This produces action sequences like:

```
[{z: 0, action: "+z_1"}, {z: 1, action: "+z_1"}, ...,
 {z: 7, action: "place", x: 0.45, y: 0.62},
 {z: 7, action: "mip"}, ...]
```

This separation also means task instances and annotation sequences can be reused with different GUI designs.

HTML-Based GUI Rendering. Each task's GUI is defined as a self-contained HTML file with embedded CSS and JavaScript. We use Playwright with headless Chromium to render screenshots. The same HTML template serves both synthetic data generation and model evaluation. During evaluation, the model's predicted clicks are executed through the GUI's JavaScript interface.

The HTML template exposes a JavaScript API for external control:

```
// Inject task data (3D points, colors, etc.)
setTaskData(sample);

// Set GUI state (current z-slice, placed markers, etc.)
setState({z: 7, mip: false, markers: [[0.45, 0.62, 7]]});

// For evaluation: execute action and get result
executeAction("+z_1");  // Returns {success: true/false}
```

Standardized Output Format. All tasks produce training data in a unified format:

- **Images**: PNG files organized as `images/{entry_idx}/{step}.png`
- **Metadata**: CSV with columns `file_name`, `action_x`, `action_y`, `action_type`, plus task-specific fields
- **HDF5 mirror**: For efficient loading, metadata is also saved as HDF5

This standardization enables a single dataset loader to train on all 9 tasks jointly.

GUISimulator for Evaluation. The same HTML-based architecture supports model evaluation. We provide a `GUISimulator` class that wraps the Playwright browser and exposes methods for interactive evaluation:

```
with GUISimulator() as sim:
    sim.load_task(sample_data)
    while not sim.is_complete():
        screenshot = sim.screenshot()
        x, y = model.predict(screenshot)
        result = sim.execute_click(x, y)  # Determines action from
            coords
    correctness = sim.check_correctness()
```

Critically, `execute_click(x, y)` determines the action type from coordinates alone, so the model cannot "cheat" by specifying action types directly. This ensures evaluation measures whether the model has learned to click the right locations, not just output valid action labels.

A.2 Main Tasks

The sequence length distribution for all tasks is depicted in Figure 6.

A.2.1 Colored Dot Tracking

The annotator traces a trajectory of colored 3D dots through z-slices. Dots are colored by position along the path (jet colormap: blue→red), derived from Omniglot handwriting strokes extruded into 3D. This task models the core challenge of connectomics proofreading: navigating a volumetric GUI to place markers in the correct order.

GUI. A 256×256 canvas shows either the current z-slice or a maximum intensity projection (MIP). Controls include z-navigation buttons (+z, -z), MIP toggle, Undo, and Done. The status bar shows current z-position and points placed.

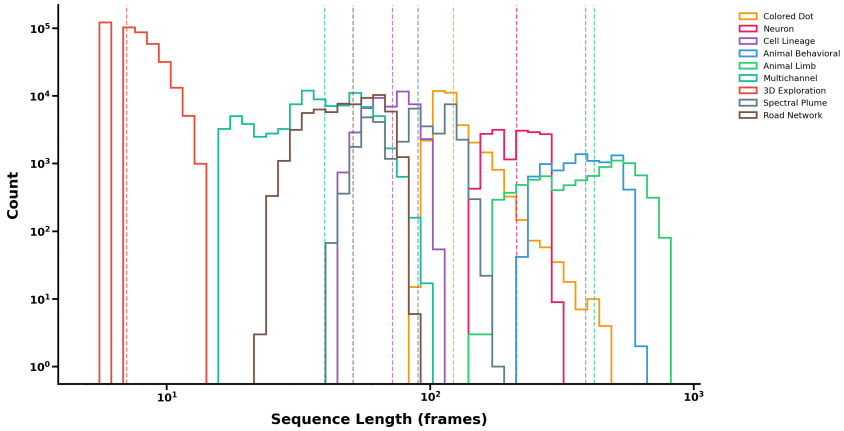


Figure 6: **Distribution of sequence lengths across the multi-task dataset.** Sequence length varies substantially across tasks, from short episodes (3D exploration classification, mean 7 actions) to long sessions (animal limb tracking, mean 420 actions; animal behavioral tracking, mean 389 actions). This variation reflects inherent differences in task complexity: 3D exploration requires only a few rotations before classification, while limb tracking demands placing many keypoints across multiple frames. The multi-task model must handle this 60× range in episode length.

Annotation Behavior. The annotator begins by toggling MIP to see the full trajectory, then navigates slice-by-slice to place markers in order. Before each placement, they verify by checking adjacent slices. With 10% probability, a point is placed incorrectly and must be undone. The sequence ends with MIP on and Done clicked.

Evaluation Metrics. Predicted and ground-truth markers are matched using the Hungarian algorithm for optimal bipartite assignment. A match is valid only if both spatial tolerances are satisfied: XY distance ≤ 0.02 in normalized coordinates (2% of the 256-pixel canvas) and Z distance ≤ 1.5 slices (out of 16 total slices). Ground-truth Z coordinates are converted from normalized [0, 1] to slice units via $z_{\text{slice}} = z \cdot 16 - 0.5$. These tolerances approximate human annotation precision. We report recall (fraction of ground-truth markers matched), precision (fraction of predicted markers that are correct), and their harmonic mean F1. For matched pairs, we additionally report RMSE_{xy} (in normalized coordinates) and MAE_z (in slice units) to quantify localization accuracy. All metrics are computed per sample and then averaged (arithmetic mean) across samples; F1 is the primary metric used for comparison plots.

A.2.2 Neuron Tracking

The annotator tracks 6–10 identical green neurons across 10 frames as the network undergoes elastic deformation, translation, and rotation. All neurons appear as identical Gaussian blobs, and identity comes purely from spatial continuity. This models calcium imaging analysis where neurons must be tracked through tissue deformation.

GUI. A 256×256 canvas shows the current frame with placed markers as labeled circles. A sidebar contains neuron ID buttons (01–10), navigation buttons (prev/next), and control buttons (cancel/undo/done).

Annotation Behavior. The annotator works through neurons in groups of 4, placing each across all frames before moving to the next group. For each neuron: select ID button, place marker, advance frame, repeat. With 10% probability, a placement is off-target and must be undone.

Evaluation Metrics. We evaluate tracking quality with three complementary metrics. A marker is considered correctly matched if its Euclidean distance to the ground-truth position is within a tolerance of 0.05 in normalized coordinates (5% of canvas size, approximately 13 pixels on the 256×256 canvas). *Track accuracy rate* measures the fraction of neurons correctly tracked across all

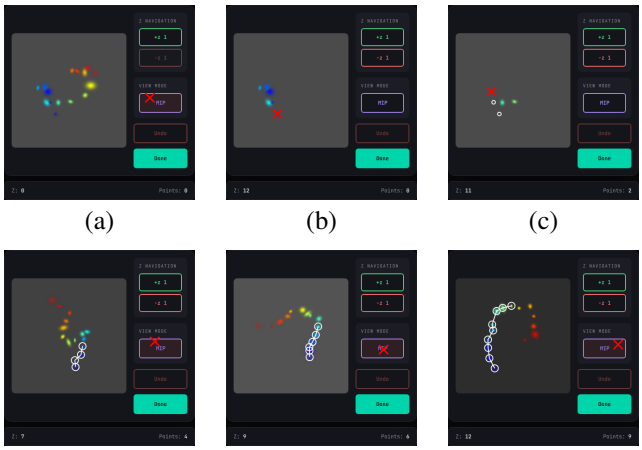


Figure 7: **Colored Dot Tracking.** Top: annotation sequence for modified hiragana from Omniglot: (a) initial MIP view showing full trajectory, (b) navigating to place first marker, (c) incorrect placement before undo. Bottom: diverse patterns (Greek, Georgian alphabets) in MIP view showing annotation progress.

frames (matched neurons / total neurons), computed per sample and then averaged across samples. *Position RMSE* quantifies localization error for markers placed outside the tolerance threshold, computed as $\sqrt{\frac{1}{N}\sum_i d_i^2}$ where d_i is the Euclidean distance (in normalized coordinates) between placed and ground-truth positions, pooled across all samples. *Detection rate* captures completeness: the fraction of neuron-frame pairs where any marker was placed (regardless of accuracy), averaged across samples. For comparison plots, we report the sample-averaged track accuracy rate as the primary metric.

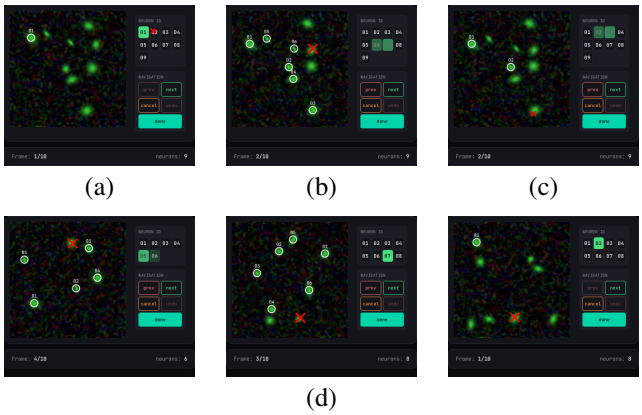


Figure 8: **Neuron Tracking.** Top: annotation sequence: (a) neuron 01 placed, selecting 02, (b) mid-group with 6 neurons placed, placing 07, (c) misclick error on background. Bottom: diversity in neuron count (6–8) and annotation progress.

A.2.3 CELL LINEAGE TRACKING

The annotator tracks cell divisions in a developing embryo rendered as Voronoi-tessellated cells within an oval boundary. Starting from a single root cell, each division requires selecting the parent marker and placing markers on both daughter cells. This models developmental biology lineage tracing where cell genealogy must be reconstructed.

GUI. A 256×256 canvas shows cells as colored Voronoi regions. Green circles mark cells in the current frame; white circles show propagated markers from previous frames. Navigation buttons (prev/next/undo/done) control the annotation.

Annotation Behavior. At frame 0, the annotator places a root marker. For each division event: navigate to the division frame, select the parent marker, then place markers on both children. Non-dividing cells auto-propagate. With 15% probability, a marker is misplaced and must be undone.

Evaluation Metrics. We represent each lineage tree as a set of directed edges (parent → child relationships) and compare the user tree E_u against ground truth E_g using exact edge matching. Edge recall measures completeness (how many true lineage relationships were captured, $|E_u \cap E_g|/|E_g|$), while edge precision measures correctness (how many annotated relationships are valid, $|E_u \cap E_g|/|E_u|$). We combine these via F1 score. Tree similarity quantifies overall structural agreement: $1 - d/(|E_u| + |E_g|)$, where d is the symmetric difference (edges that must be added or removed to match). Division accuracy specifically evaluates cell division events, measuring the fraction of true divisions (parent splitting into exactly two children) that were correctly identified with the correct parent-child assignments. All metrics are computed per sample and averaged across samples; we report the mean edge F1 score as the primary metric for comparisons.

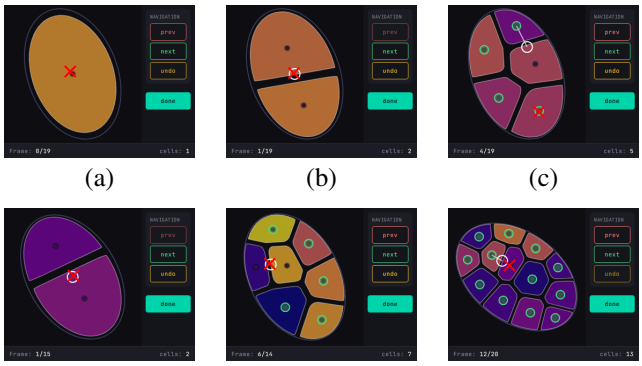


Figure 9: **Cell Lineage Tracking.** Top: annotation sequence: (a) placing root marker on single cell, (b) selecting parent after first division, (c) misclick on wrong cell. Bottom (d): progression from 2 cells to 13 cells showing increasing lineage complexity.

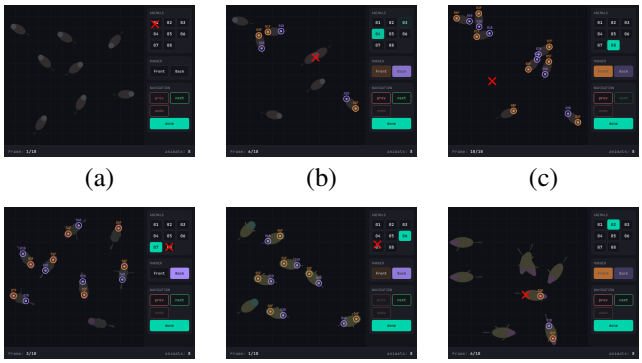
A.2.4 ANIMAL BEHAVIORAL TRACKING

The annotator tracks 4–8 animals across 10 video frames by placing front (head) and back (tail) markers on each. Animals have variable morphology (body segments, limbs, head shapes) but consistent appearance within a sequence. This models behavioral tracking studies where position and orientation must be annotated.

GUI. A 384×384 canvas shows the current frame. A sidebar contains animal selection buttons (01–08), marker type buttons (Front/Back), and navigation controls. Markers appear as labeled circles (e.g., "01F", "02B").

Annotation Behavior. The annotator works through animals in order: select animal, place front marker on head, place back marker on tail, repeat for all animals, then advance frame. With 10% probability each for near errors (slight offset) and far errors (random misclick), followed by undo.

Evaluation Metrics. We evaluate three aspects of annotation quality. *Animal match rate* measures tracking accuracy: for each sample, we compute the fraction of correctly placed marker pairs (front and back within 5% of the canvas, i.e., $\sim$19 pixels for the 384-pixel canvas) out of all expected pairs (animals $\times$ frames), then average across samples. *Position RMSE* quantifies localization precision by computing root-mean-square error of position offsets for incorrectly placed markers, on a normalized $[0, 1]$ scale. *Detection rate* captures completeness: the fraction of samples where all required markers were placed (i.e., no missing annotations).



(a) (b) (c)

(d)

Figure 10: **Animal Behavioral Tracking.** Top: annotation sequence: (a) selecting first animal, (b) mid-annotation with 3 animals marked on frame 6, (c) far error misclick near completion. Bottom: morphological diversity: insects with antennae, fish-like bodies, elongated forms.

A.2.5 Animal Limb Tracking

The annotator places keypoint markers on animal body parts across video frames. Animals (birds, insects, spiders, snakes) are rendered as shapes without visible keypoint markers, so positions must be inferred from morphology. This models pose estimation annotation where keypoints are implicit rather than marked.

GUI. A 256×256 canvas shows the current frame. A tabbed sidebar organizes keypoints by body part (Body, Legs, Wings, etc.), with buttons for each keypoint. Navigation buttons (prev/next) switch frames. The status bar shows frame number and keypoints placed.

Annotation Behavior. The annotator works through tabs in order, clicking each keypoint button then placing it on the rendered animal. Limbs articulate $\sim$70° between frames, requiring re-inference of positions. With 10% probability, a keypoint is placed slightly off (near error) and must be undone.

Evaluation Metrics. We evaluate keypoint localization using a Percentage of Correct Keypoints (PCK) proxy: the fraction of placed keypoints that fall within 3 pixels of their ground-truth positions. For each sample, we compute per-sample keypoint accuracy as correct_keypoints/total_keypoints, then report the arithmetic mean across all samples as the final keypoint accuracy metric used in plots. This measures spatial precision, specifically whether the model correctly infers body part locations from rendered morphology. We also track completion rate (whether the model signals task completion by clicking "done" after placing keypoints) to assess task understanding. The partial credit score combines both aspects: completed tasks receive full PCK credit, while incomplete tasks receive half credit ($0.5 \times$ PCK), penalizing models that place accurate keypoints but fail to recognize when annotation is finished.

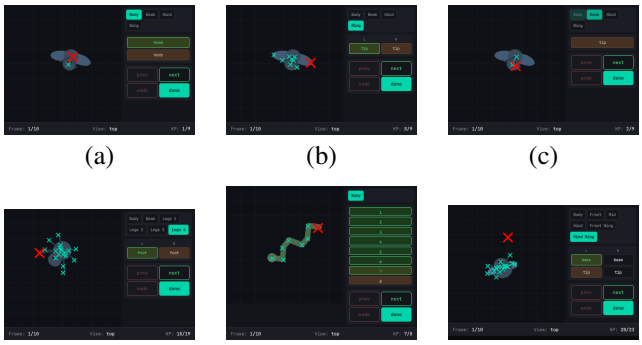


Figure 11: **Animal Limb Tracking.** Top: bird annotation sequence: (a) early annotation with Body tab, (b) later with Wing tab and 8/9 keypoints placed, (c) near-error on beak before undo. Bottom: morphological diversity: spider with 8 legs, snake with spine-only keypoints, flying insect with wings.

A.2.6 Multichannel Image Alignment

The annotator places corresponding landmark points across multiple image channels that have spatial misalignment. Each channel shows the same underlying scene with different appearance and geometric distortion.

Data Sources. 50% synthetic images (5 DGPs: Gaussian blobs, Perlin noise, geometric shapes, Voronoi cells, gradient fields) and 50% natural images (ImageNette with multichannel projection). Each instance has 3–6 channels with 3–9 landmark points.

GUI. The interface displays a 256×256 image (current channel) with channel selector dots and point buttons. Placed markers show as numbered circles. Status bar indicates current channel, point index, and completion count.

Annotation Behavior. For each landmark: click to place on current channel, switch to next channel, repeat across all channels. First 3 points are positioned for optimal affine estimation. 15% error rate with undo correction; 1px click noise.

Evaluation Metrics. We evaluate alignment quality via the match rate, which measures the fraction of annotated landmarks whose positions fall within 2 pixels of the ground truth across all channels ($n_{\text{matched}}/n_{\text{annotated}}$). To assess task progress, we report the completion rate $\min(1, n_{\text{landmarks}}/3)$, reflecting that at least 3 corresponding landmark pairs are needed to estimate an affine transformation (6 degrees of freedom). We also track the total annotation count as a measure of annotator engagement. For plotting and model comparison, we report the mean match rate averaged across all evaluation samples.

A.2.7 Spectral Plume Finding

The annotator identifies real plumes among confounders by checking their presence across multiple spectral bands, then draws their boundaries. Real plumes appear in exactly 3 of 5 bands; confounders appear in 1, 2, or all 5 bands.

Data. Synthetic scenes with 1–3 real plumes and 3–6 confounders per 256×256 image. Objects are organic blob shapes with soft edges. Background uses aerial/satellite imagery textures.

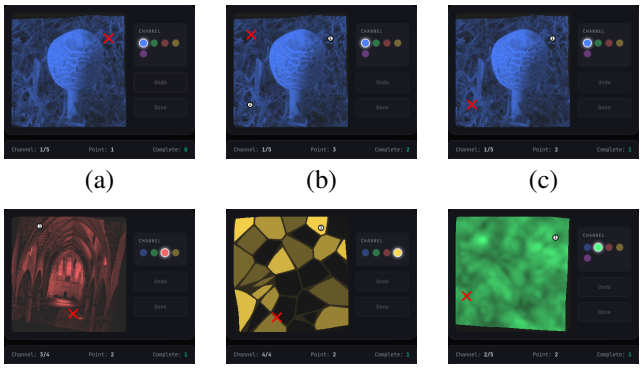


Figure 12: **Multichannel Image Alignment.** Top: annotation sequence on a natural image (mushroom): (a) initial landmark placement, (b) mid-progress with 2 points complete, (c) wrong click location. Bottom: data diversity: church interior (natural), Voronoi cells (synthetic), Perlin noise (synthetic).

GUI. The interface shows the current spectral band with toggle buttons (b1–b5) to switch views. Markers can be placed to track objects across bands. A Draw button enters polygon mode for boundary annotation; Cancel removes ongoing polygons.

Annotation Behavior. (1) Explore: toggle through bands to identify objects and count their band appearances. (2) Place markers on candidate plumes. (3) Draw: click polygon vertices around each verified plume; click near first point to close. (4) Done when all plumes are outlined.

Evaluation Metrics. We evaluate both detection quality and segmentation accuracy. Predicted polygons are matched to ground-truth plume boundaries using greedy matching with an IoU threshold of 0.3: we compute an IoU matrix between all predicted and ground-truth polygons, sort pairs by IoU descending, and greedily assign matches above the threshold. Detection recall measures the fraction of true plumes that were found, capped at 1 to avoid inflating scores when excess predictions match the same target. Detection precision measures the fraction of predicted plumes that correspond to real plumes (number of matched polygons / number of predictions). Count accuracy penalizes over- or under-counting: $\max(0, 1 - |n_{\text{found}} - n_{\text{expected}}| / \max(n_{\text{expected}}, 3))$. Mean IoU averages the polygon overlap scores for all matched plumes, capturing segmentation quality. All metrics are computed per sample and averaged across samples for reporting. The combined score balances detection and segmentation: $0.5 \cdot F_1 + 0.5 \cdot$ mean IoU, where F_1 is the harmonic mean of recall and precision. This combined score is the primary metric used for comparisons.

A.2.8 Road Network Construction

The annotator traces road networks by placing nodes at intersections and connecting them with edges. The goal is to reconstruct the graph topology of visible road segments in aerial imagery.

Data. Aerial/satellite images showing road networks with varying complexity. Each instance has ground truth node positions (intersections, endpoints) and edge connectivity. Images are 256×256 with roads highlighted by lane markings.

GUI. The interface overlays the road image with interactive node/edge creation. Clicking places a node; clicking an existing node in "Connect to..." mode creates an edge. Undo removes the last action; Remove deletes selected elements. Status shows node and edge counts.

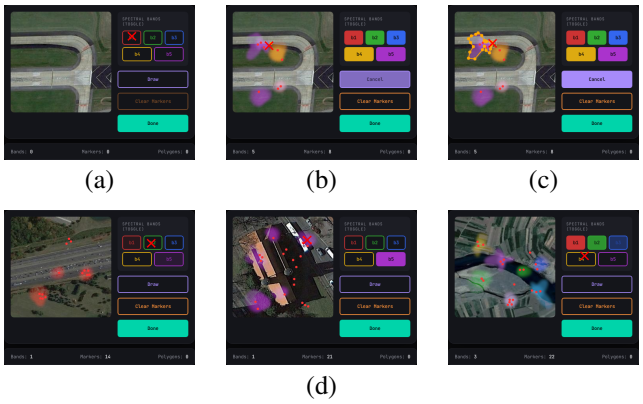


Figure 13: **Spectral Plume Finding.** Top: annotation sequence: (a) initial band toggle, (b) drawing mode with 5 bands toggled and 8 markers, (c) polygon outline nearly complete. Bottom: scene diversity: highway intersection, building/parking area, agricultural fields.

Annotation Behavior. The annotator places nodes at road intersections and endpoints, then connects them with edges following visible road segments. Invalid placements (off-road, duplicate nodes) trigger immediate undo. Edges are created by selecting source and target nodes.

Evaluation Metrics. We evaluate both node placement and edge connectivity. Predicted nodes are matched to ground truth nodes using the Hungarian algorithm with a 6-pixel tolerance, yielding node recall (fraction of GT nodes found) and precision (fraction of predictions that are correct). For edges, we map each predicted edge through the node matching: an edge counts as correct only if both endpoints matched valid GT nodes and the corresponding GT edge exists. This captures whether the model correctly identified the road network's topological structure, not just node locations. We compute F1 scores for both nodes and edges. For each sample, Graph F1 = (Node F1 + Edge F1) / 2; we then average Graph F1 across samples for plotting.

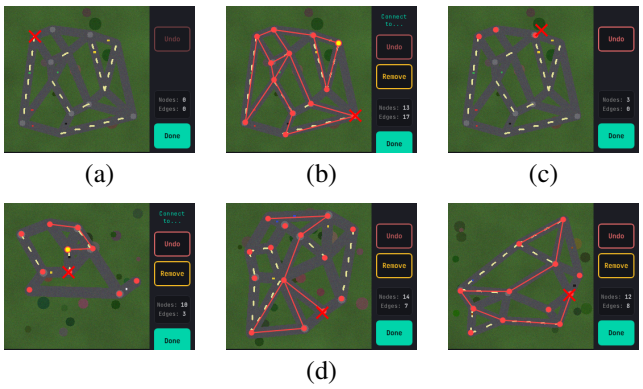


Figure 14: **Road Network Construction.** Top: annotation sequence: (a) placing first node, (b) near completion with 13 nodes and 17 edges, (c) invalid node placement. Bottom: diverse road layouts with varying complexity (10–14 nodes, 3–8 edges).

A.2.9 3D EXPLORATION CLASSIFICATION

The annotator rotates a 3D object to explore it from multiple viewpoints, then classifies it into one of 9 classes. All objects consist of 5 colored spheres (red, blue, green, orange, purple) arranged in distinct configurations. From certain viewing angles, different arrangements appear similar due to occlusion, so rotation is required to reveal the true 3D structure.

Object Classes. The 9 classes divide into 5 planar and 4 truly three-dimensional arrangements:

#	Class	Description	3D?
1	Line	5 collinear spheres	No
2	Plus	4 arms + 1 center (cross)	No
3	Pentagon	5 in regular ring	No
4	T-shape	3 across + 2 stem	No
5	L-shape	3 horizontal + 2 vertical	No
6	Methane	1 center + 4 tetrahedral	Yes
7	Square pyramid	4 base + 1 apex	Yes
8	Trigonal bipyramid	3 equatorial + 2 poles	Yes
9	Bowtie	2 triangles sharing vertex	Yes

Table 2: **3D Object Classes.** The 9 object classes for 3D exploration classification.

Each class has a distinct topology (neighbor relationships), making them distinguishable given sufficient viewpoints. The 4 truly 3D classes require rotation to classify correctly.

GUI. The interface displays a 256×256 Three.js canvas showing the 3D object. A control panel provides rotation buttons (+X, -X, +Y, -Y, each rotating 30°) and 9 class buttons. Objects start at a random orientation (uniform over SO(3)). A confirm button finalizes the selection.

Annotation Behavior. The virtual annotator performs 3–10 rotations (mean 5, Gaussian distribution) to explore the object, avoiding consecutive presses of the same button. After exploration, they select a class button. With 30% probability, a misclick occurs (wrong class selected) followed by correction. Finally, confirm is clicked.

Evaluation Metrics. We evaluate four aspects of performance. *Flat vs. 3D accuracy* measures whether the model correctly identifies if a shape is planar (classes I, +, P, T, L) or truly three-dimensional (classes M, A, Y, X), a coarse but fundamental distinction. *Class similarity* awards partial credit for near-misses: 1.0 for exact matches, 0.3 for confusions within the same group (e.g., mistaking one flat shape for another flat shape, or one 3D shape for another 3D shape), and 0 for cross-group errors. *Exploration quality* captures how thoroughly the model examines the object before classifying, computed as $\min(r/5, 1)$ where r is the number of rotations performed (saturating at 5 rotations). Finally, *exploration-adjusted accuracy* rewards correct classifications in proportion to their exploration quality: for correct predictions it equals $\min(r/5, 1)$, for incorrect predictions it equals 0. All metrics are computed per sample and then averaged (simple mean) across all samples; we report the mean exploration-adjusted accuracy in plots.

A.3 OOD TASK: SHAPE MATCHING

The annotator clicks all objects matching a template shape shown in the sidebar. This task tests generalization to an unseen task format not included in training.

Data. Each instance contains 12 randomly placed objects from 4 shape classes (circle, triangle, square, star) in various colors. A template shape is shown in the sidebar; 2–5 objects match it. Objects are placed on a dark background without overlap.

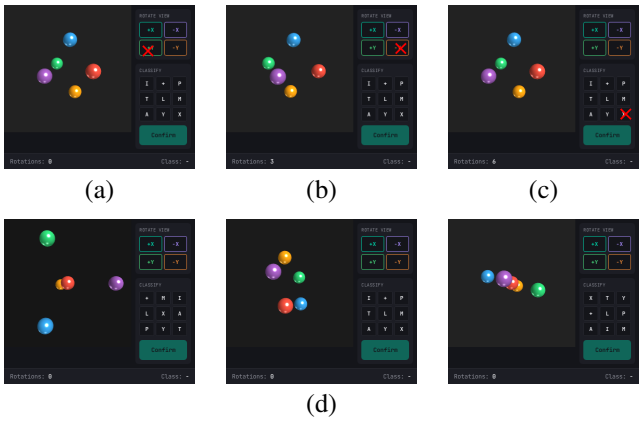


Figure 15: **3D Exploration Classification.** Top: annotation sequence for square pyramid: (a) initial random orientation, (b) after several rotations exploring structure, (c) misclick on wrong class button. Bottom: class diversity: methane (truly 3D), pentagon (flat), plus/cross (flat).

GUI. The interface shows a "MATCH THIS" sidebar with the template shape and a "FOUND: X/N" counter tracking progress. Clicked matching objects display a green checkmark. A Done button completes the annotation.

Annotation Behavior. The annotator scans the scene and clicks each object matching the template. The counter increments with each correct click. After finding all targets, Done is clicked.

Evaluation Metrics. We evaluate shape matching performance using recall and precision-style metrics. A click registers as hitting an object if it falls within 17 pixels of the object center (object radius of 12 pixels plus 5-pixel tolerance). The all-targets rate measures the fraction of instances where every matching object was successfully clicked, assessing completeness of visual search. The no-extras rate measures the fraction of instances where no non-matching objects were clicked, assessing discrimination accuracy. Full correctness requires both: all targets clicked and no extras, meaning the model must both find all matching shapes and avoid false positives. We also track completion rate (whether Done was clicked) to assess understanding of task termination. For plots, we report the all-targets rate averaged across all test instances.

B METHODOLOGY

B.1 MODEL

We use a Vision-Language Model (VLM) that processes interleaved sequences of GUI screenshots and click coordinates. The architecture consists of a DINOv2 vision encoder and a transformer head.

Vision Encoder. We use DINOv2 with registers (Darcet et al., 2023) as the vision backbone. All encoder weights are unfrozen during training. Images are processed at native resolution with patch size 14, then spatially pooled to a fixed grid of $12 \times 9 = 108$ tokens per image via adaptive average pooling. Learnable position embeddings are added to the pooled features.

Transformer Head. The transformer head processes the interleaved sequence of image tokens and click tokens. Each block uses multi-head self-attention with RoPE positional encoding (base frequency 10000), layer normalization (pre-norm), MLP with expansion ratio 4.0 and GELU activation, and dropout 0.1. See Table 3 for size configurations. Attention is *block-causal*: bidirectional within each image's patch tokens, but causal across the sequence (each frame can only attend to previous frames). See Figure 17 for an illustration.

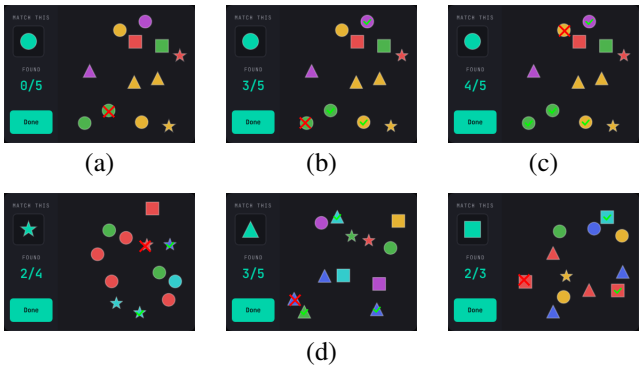


Figure 16: **Shape Matching (OOD).** Top: annotation sequence for circle template: (a) initial state (0/5), (b) mid-progress (3/5), (c) near completion (4/5). Bottom: template diversity: star (2/4), triangle (3/5), square (2/3).

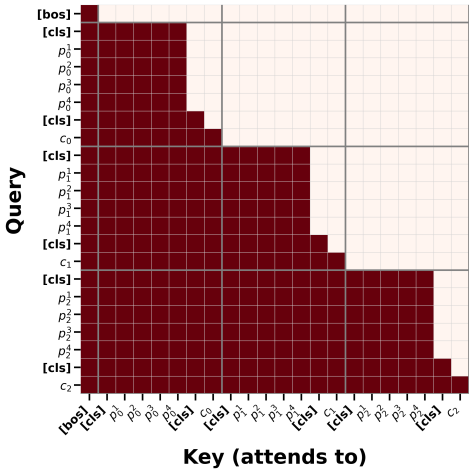


Figure 17: **Block-Causal Attention Mask.** Illustration with 3 patches per frame; the actual model uses 108). Blue indicates allowed attention. Within each frame, all tokens attend bidirectionally. Across frames, attention is causal: Frame 1 attends to Frame 0, but not vice versa.

Sequence Format. The input sequence has the following structure:

$$\texttt{[bos]}\ \texttt{[cls]}\ \texttt{img}_0\ \texttt{[cls]}\ \texttt{click}_0\ \texttt{[cls]}\ \texttt{img}_1\ \texttt{[cls]}\ \texttt{click}_1\ \cdots\ \texttt{[cls]}\ \texttt{img}_t\ \texttt{[cls]}$$

where img_i denotes the 108 patch tokens for frame i, and click_i is a single token encoding the (x, y) coordinate. Modality embeddings distinguish token types: `cls`(0), `image`(1), `click`(2), `bos`(3). The context length is 20 frames.

Coordinate Prediction. Tasks have different native image sizes (ranging from 336×258 to 524×436 pixels). For multi-task training, all images are resized to the maximum dimensions (524×436). Coordinates are stored as relative values in $[0, 1]$ and discretized into bins matching the maximum pixel dimensions: 524 bins for x and 436 bins for y. This provides pixel-accurate prediction on the resized images. Input coordinates use separate learned embeddings for x and y, which are concatenated and projected to the model dimension. Output coordinates are predicted via

separate linear heads:

$$\hat{x} = \text{Linear}_x(h) \in \mathbb{R}^{524} \tag{1}$$

$$\hat{y} = \text{Linear}_y(h) \in \mathbb{R}^{436} \tag{2}$$

where h is the hidden state at the `[cls]` token preceding the target click. The loss is the sum of cross-entropy losses over x and y.

Model Sizes. We train four model sizes with log-linear parameter progression. Table 3 shows the configurations.

Backbone	Dim	Heads	Layers	Params
ViT-S/14	384	6	12	22M
ViT-B/14	768	12	12	86M
ViT-L/14	1024	16	24	304M

(a) **DINOv2 Vision Encoder.** With registers. Patch size 14, MLP ratio 4.0.

Size	Backbone	Dim	Layers	Heads	Total
Very Small	ViT-S/14	256	4	4	25M
Small	ViT-S/14	384	6	6	28M
Base	ViT-B/14	512	8	8	95M
Large	ViT-L/14	768	12	12	320M

(b) **Transformer Head.** MLP ratio 4.0, dropout 0.1.

Table 3: **Model Configurations.** (a) DINOv2 backbone specifications. (b) Transformer head and total parameter counts.

B.2 TRAINING

Hyperparameter	Value
Optimizer	AdamW
Weight decay	0.01
Betas	(0.9, 0.999)
Learning rate (head)	10^{-4}
Learning rate (backbone)	10^{-5} (10× reduction)
Gradient clipping	max norm 1.0
LR schedule	Linear warmup + cosine decay to 10%
Warmup steps	4000 (very small, small), 2000 (base), 1000 (large)
Batch size per GPU	4
Gradient accumulation	2 steps
GPUs	4× NVIDIA A100 80GB
Effective batch size	32

Table 4: **Training Hyperparameters.**

Training Steps. To compare models at equal compute (Kaplan et al., 2020; Hoffmann et al., 2022), we scale training steps inversely with model size:

Size	Params	Steps	Checkpoints
Very Small	25M	380,000	every 2,000
Small	28M	300,000	every 2,000
Base	95M	100,000	every 1,000
Large	320M	30,000	every 500

Table 5: **FLOP-Matched Training Configurations.** Steps are scaled so that total compute (params × steps) is approximately constant.

We also compare models at equal loss to test whether larger models are more data-efficient (Section 4.2).

Single-Task vs. Multi-Task. For single-task experiments, we train the base model on Colored Dot Tracking for one epoch (109,355 steps), processing approximately 3.5M context windows (70M images at 20 frames per window). For multi-task experiments, we train on all 9 tasks jointly with uniform sampling. The combined multi-task dataset contains 21.8M context windows; approximately 2.4% of samples use padding tokens for shorter sequences (primarily 3D Exploration Classification). Multi-task models are trained for varying steps depending on model size (Table 5), ranging from 960K samples (Large) to 12.2M samples (Very Small). The training procedure is otherwise identical.

B.3 Evaluation

We evaluate models in two modes: *teacher-forced* and *autoregressive*. Task-specific evaluation metrics are described alongside each task in Appendix A.2.

Teacher-Forced Evaluation. Given ground-truth action history, the model predicts the next action. We measure per-action-type accuracy (e.g., button clicks vs. canvas placements) and canvas placement error at multiple thresholds (1px, 3px, 5px, 10px). This provides fine-grained learning dynamics without compounding errors.

Autoregressive Evaluation. The model interacts with the GUISimulator in a closed loop, predicting clicks from screenshots until it clicks "Done" or reaches a maximum step limit ($2\times$ ground-truth length). The simulator executes each click through the GUI's JavaScript interface, determining action type from coordinates alone, so the model cannot specify action types directly.

Inference Parameters.

- Temperature: 0.4
- No beam search
- Context window: 20 frames (same as training)

Strict vs. Generous Metrics. Binary "is_correct" evaluation is harsh: partial progress receives zero credit. We additionally compute *generous metrics* using Hungarian matching to optimally align predicted and ground-truth outputs, then compute precision, recall, F1, and position errors on matched pairs. This enables finer-grained analysis of model capability. See each task's Evaluation Metrics paragraph in Appendix A.2 for task-specific definitions.

Evaluation Sets. For teacher-forced evaluation, we sample 1024 random 20-frame windows from test sequences. For autoregressive evaluation, we run 50–300 complete episodes per checkpoint (50 for training dynamics, 300 for final evaluation). Confidence intervals are computed via binomial standard deviation for accuracy metrics.

B.4 ICL & Fine-Tuning

We evaluate whether multi-task pretraining enables adaptation to a held-out task (Shape Matching, Appendix A.3) through in-context learning or fine-tuning. All experiments use the base model checkpoint at 100,000 steps (95M parameters).

In-Context Learning Protocols. We test three ICL conditions, all evaluated on 256 test instances with temperature 0.4 and maximum 50 generation steps:

- **Zero-shot (ZS)**: The model receives only the first frame of a test instance and generates subsequent actions autoregressively. This tests whether the model can infer the task from the visual template alone.
- **Prefix**: The model receives the first 2 ground-truth frame-action pairs from a test sequence, then generates the remaining actions. This tests within-sequence continuation.
- **Few-shot (FS)**: Three complete demonstration sequences from the training set are prepended to the context before the test instance's first frame. Since Shape Matching sequences are short (5–8

steps), three demos fit within the 20-frame context window. This tests true in-context learning from examples.

All three conditions achieve negligible accuracy ($<2\%$), indicating that multi-task pretraining does not induce in-context learning capabilities for novel annotation tasks.

Fine-Tuning Protocol. We fine-tune from the pretrained checkpoint using full fine-tuning with differential learning rates: 10^{-4} for the transformer head and 10^{-5} for the DINOv2 backbone ($0.1\times$ scaling). Other hyperparameters: 500 warmup steps, cosine decay, weight decay 0.01, gradient clipping at 1.0. We train for 1,000 steps on subsets of varying size (50, 100, 150, 250, 350, 500 sequences) and 5,000 steps on the full dataset (7,800 sequences).

Fine-tuning achieves 76.6% accuracy with just 500 sequences, saturating near this level; 7,800 sequences yields similar performance (77.7%). See Figures 24 and 25 for training dynamics and data efficiency curves.

From-Scratch Baseline. To isolate the contribution of pretraining, we train the identical architecture from random initialization on the Shape Matching task. We test two learning rate configurations: an aggressive setting (3×10^{-4}, 2000 warmup steps) and a matched setting (same as fine-tuning). Both are trained for 5,000 steps on 7,800 sequences.

Training from scratch fails entirely (0% accuracy) regardless of hyperparameters, demonstrating that multi-task pretraining provides essential inductive biases for GUI annotation that cannot be recovered through extended training on the target task alone.

B.5 Model Internal Analysis

We probe the internal representations of the trained multi-task model to understand what information is encoded during annotation. All experiments use the base model checkpoint at 100,000 steps (95M parameters), extracting activations from layer 6 of 8 (75% depth, 512 dimensions) at the [cls] token position preceding each action prediction.

Activation Extraction. For each task, we sample 100 test sequences and perform a forward pass through the model, hooking the transformer layer to extract hidden states. Each action yields a 512-dimensional activation vector paired with metadata (frame index, action type, mistake labels, task-specific state). Activations are saved in HDF5 format for efficient downstream analysis.

Linear Probe Training. All probes use the same methodology: activations are standardized (zero mean, unit variance), split 80/20 into train/test sets (stratified for classification), and fit with either logistic regression (classification; $C = 1.0$, balanced class weights) or ridge regression (continuous labels; $\alpha = 1.0$). We report test-set metrics and 5-fold cross-validation scores.

Mistake and Correction Probes (Figure 5a). We define "mistake" as an action immediately preceding an undo, and "correction" as the undo action itself. For the pooled analysis across all 9 tasks, we train a single logistic regression probe on 130k actions ($\sim$4.4% mistakes). The mistake probe achieves ROC AUC = 0.87 (CV: 0.83 ± 0.07), while the correction probe achieves ROC AUC = 0.99 (CV: 0.96 ± 0.04). The higher correction accuracy reflects that corrections have distinctive activation patterns (reversal actions with consistent structure), while mistakes are more heterogeneous.

Diverse Probes (Figure 5b). We train probes for multiple latent concepts across tasks:

- **Task phase** (road_network): Binary classification of node placement vs. edge construction phases achieves ROC AUC = 1.00, indicating the model strongly encodes workflow stage.
- **Object class** (3d_exploration): 9-way classification of 3D object type achieves ROC AUC = 0.93, suggesting the model identifies object geometry during exploration.
- **Task progress**: Regression on normalized progress (fraction of annotation completed) achieves $R^2 = 0.65$–0.92 depending on task, with colored_dot_tracking showing strongest signal ($R^2 = 0.92$ for color progress).
- **Error states**: Per-task mistake probes achieve ROC AUC = 0.83–0.91 across tasks.

- **Temporal position**: Frame index classification achieves ROC AUC = 0.70–0.89 across tracking tasks, with cell_lineage_tracking showing best frame encoding (0.89).

Cross-Task PCA (Figure 5c). We pool activations from all 9 tasks, standardize, and compute 2-component PCA. Mistakes (actions before undo) form a partially separable cluster from correct actions, suggesting a shared "something is wrong" representation. However, task identity also clusters in activation space, indicating both task-specific and task-general structure.

Leave-One-Task-Out Transfer (Figure 5d). To quantify generalization of mistake representations, we train probes on 8 tasks and test on the held-out task. Mean transfer ROC AUC = 0.71 ($\pm$0.16), with 8/9 tasks showing above-chance transfer. The outlier is 3d_exploration_classification (ROC AUC = 0.29), which has fundamentally different error structure (classification mistakes vs. spatial placement errors). This suggests mistake detection is partially universal but retains task-specific components.

Probe Direction Analysis. We analyze cosine similarity between per-task probe coefficient vectors. Off-diagonal similarity ranges from 0.35–0.50, indicating moderate alignment of mistake directions across tasks. Some task pairs show stronger alignment (animal_behavioral $\leftrightarrow$ animal_limb $\leftrightarrow$ road_network: similarity 0.6–0.7), while others are more isolated (colored_dot_tracking, 3d_exploration).

C Additional Results

C.1 Single-Task Training

We analyze training dynamics on a single task (Colored Dot Tracking) to understand skill acquisition in isolation. Figure 18 shows how action accuracy evolves during training under teacher-forced evaluation, alongside skill accuracy under generative evaluation. Figure 19 provides additional generative metrics including match quality, RMSE, episode length, action distributions, and click heatmaps.

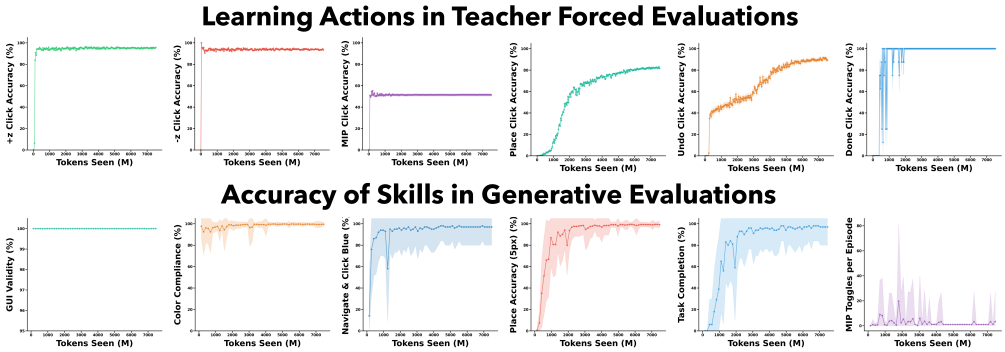


Figure 18: **Single-Task Training Dynamics.** Top: Accuracy of various actions (as assessed by teacher-forced evaluation) in single-task colored dot tracking. Shaded area denotes 1 standard deviation. Bottom: Accuracy of various skills (as assessed by generative evaluation) in single-task colored dot tracking. The shaded area denotes 1 standard deviation.

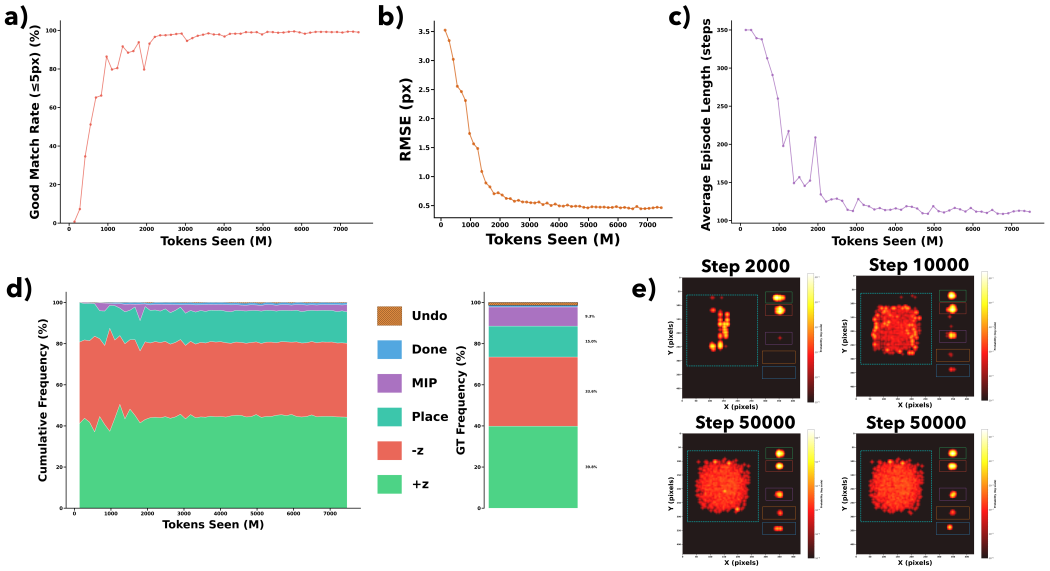


Figure 19: **Additional Generative Evaluation Results.** Results for single-task colored dot tracking. (a) Matches are determined via a Hungarian matching procedure between model annotations and the ground truth annotations. A good match is such that it is within 5 px of the ground truth annotation. (b) RMSE of good matches. (c) Episode length decreases over training. (d) Frequency of various actions during generative evaluation compared to ground truth distribution. (e) Heatmap of model clicks throughout training.

C.2 MULTI-TASK SCALING

We study the training dynamics of downstream decision and motor task metrics (Figure 20). We examine how performance scales with model size and compute in the multi-task setting. Figure 21 shows training loss as a function of FLOPs across model scales, revealing diminishing returns from increased parameters. Figure 22 isolates spatial precision by measuring placement accuracy within 5 pixels, showing inverse scaling for some tasks. Figure 23 reports task-specific metrics across all 9 tasks, demonstrating that larger models do not consistently improve performance.

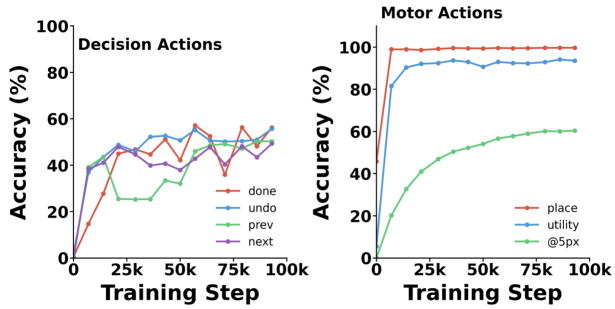


Figure 20: **Decision vs Motor Metrics.** Training dynamics of downstream task metrics. (a) Decision action classification accuracy for done, undo, prev, and next actions. (b) Motor metrics including place action accuracy, utility action accuracy, and placement precision within 5 pixels.

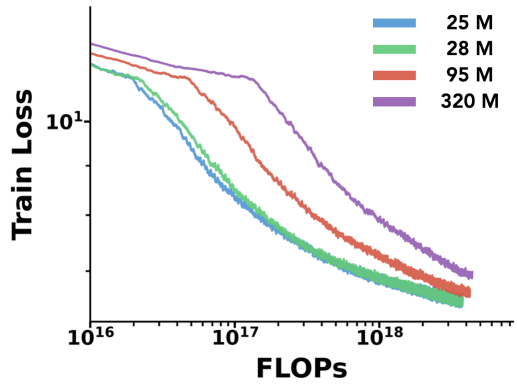


Figure 21: **Scaling with Compute.** Training loss as a function of compute across model scales. Loss (log scale) versus cumulative training FLOPs (log scale) for four model sizes: Very Small (25M), Small (28M), Base (95M), and Large (320M parameters). The plot is cropped to FLOPs $\geq 10^{16}$ to focus on the converged training regime. All models follow similar loss trajectories when plotted against compute, with larger models achieving marginally lower loss at equivalent FLOPs. The overlapping curves suggest that within this model size range, scaling up parameters provides diminishing returns relative to simply training smaller models longer, consistent with the finding that downstream task performance does not improve monotonically with model size.

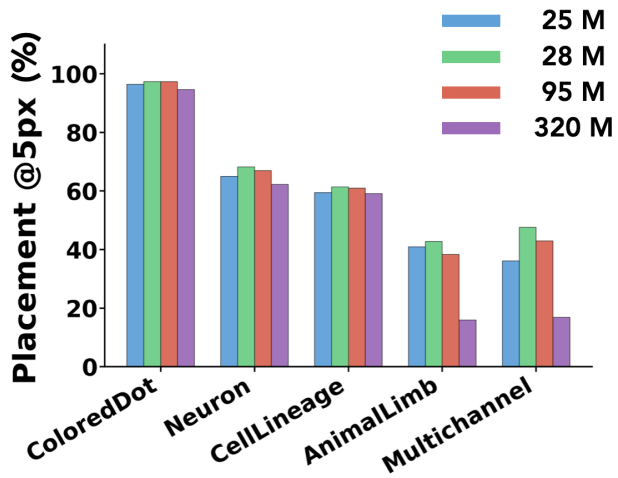


Figure 22: **Placement Precision Across Model Scales.** Fraction of placement actions landing within 5 pixels of the ground truth target under teacher-forced evaluation, grouped by task and model size. This metric isolates spatial precision from action selection by evaluating only placement actions. Colored Dot Tracking achieves consistently high precision (94–97%) across all scales, while Neuron Tracking and Cell Lineage show moderate precision (59–68%). Notably, placement accuracy degrades substantially for the Large model on Animal Limb (40.9% $\rightarrow$ 16.0%) and Image Alignment (47.6% $\rightarrow$ 16.9%), resulting in overall placement precision dropping from 63.5% (Small) to 49.8% (Large). This inverse scaling suggests that larger models may overfit to action-type prediction at the expense of fine-grained spatial localization.

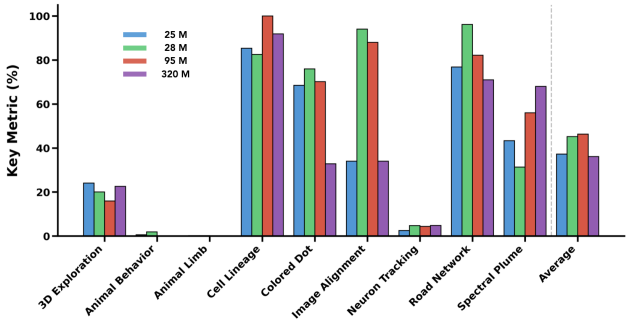


Figure 23: **Task-Specific Performance Across Model Scales.** Key metrics for each of the 9 tasks under autoregressive evaluation on models of increasing size (Very Small: 25M, Small: 28M, Base: 95M, Large: 320M parameters). Each task uses its most informative metric: F1 scores for tracking tasks, accuracy for classification, and completion rate for alignment. Performance varies substantially across tasks, with Cell Lineage achieving 82–100% and Road Network reaching 71–96%, while Animal Behavior and Animal Limb remain below 2% across all scales. Notably, scaling does not consistently improve performance: the Large model underperforms smaller models on several tasks (Colored Dot, Image Alignment, Road Network), and average performance peaks at the Base model (46.3%) before declining at Large (36.1%).

C.3 Downstream Adaptation

We provide additional analysis of fine-tuning behavior on the out-of-distribution Shape Matching task. Figure 24 compares training duration and dataset size effects, and demonstrates that training from scratch fails entirely while fine-tuning succeeds. Figure 25 shows loss curves across dataset sizes, illustrating the divergence between training loss and downstream accuracy.

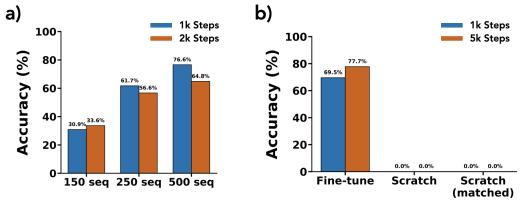


Figure 24: **Fine-Tuning Variants.** (a) Comparison of 1,000 vs 2,000 training steps across different dataset sizes. Longer training improves performance with limited data (150 sequences) but causes overfitting with larger datasets, with accuracy dropping from 76.6% to 64.8% for 500 sequences. (b) Fine-tuning from a pretrained checkpoint versus training from scratch, evaluated at 1,000 and 5,000 steps. Fine-tuning achieves 69.5% accuracy after just 1,000 steps and 77.7% after 5,000 steps, while training from scratch fails completely (0% accuracy) regardless of training duration, demonstrating that multitask pretraining provides essential inductive biases that cannot be recovered through extended training alone.

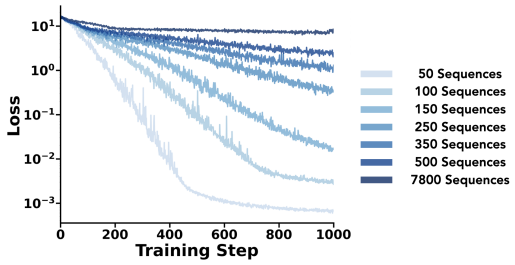


Figure 25: **Fine-Tuning Loss Curves.** Training loss (log scale) over the first 1,000 steps for models fine-tuned on varying amounts of data, from 50 to 7,800 sequences. Smaller datasets (lighter colors) exhibit faster loss reduction and reach lower final loss values, indicating rapid overfitting to the limited training data. Larger datasets (darker colors) show more gradual loss decay, consistent with better generalization. The divergence between training loss and downstream accuracy (where 500 sequences outperforms smaller datasets despite higher loss) highlights that lower training loss does not necessarily correspond to better task performance.

C.4 TRAINING ABLATIONS

We report ablation studies on key training hyperparameters and design choices. Figure 26 compares learning rates for the base model. Figure 27 evaluates DINOv2 variants and training strategies on single-task performance. Figure 28 shows training loss versus wall-clock time across model sizes. Figure 29 compares sliding-window versus non-overlapping context window construction for data augmentation.

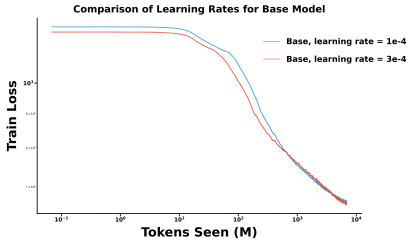


Figure 26: **Learning Rate Comparison.** Train loss for the base model variant across learning rates.

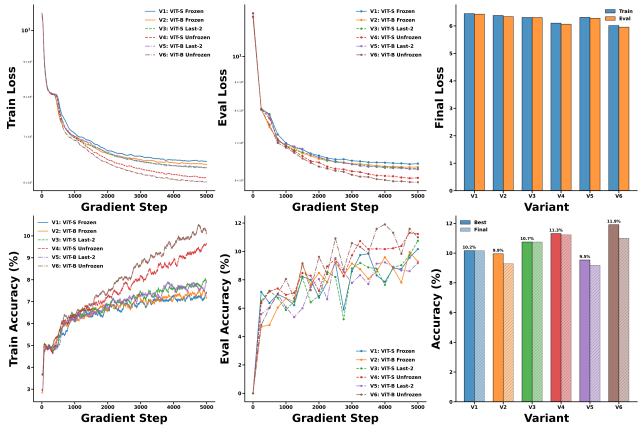


Figure 27: **DINOv2 Training Strategies.** Comparing model sizes and training approaches on colored dot tracking.

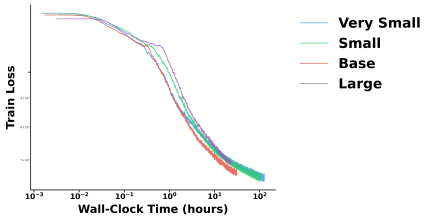


Figure 28: **Training Time Analysis.** Train loss as a function of wall-clock time across four model sizes.

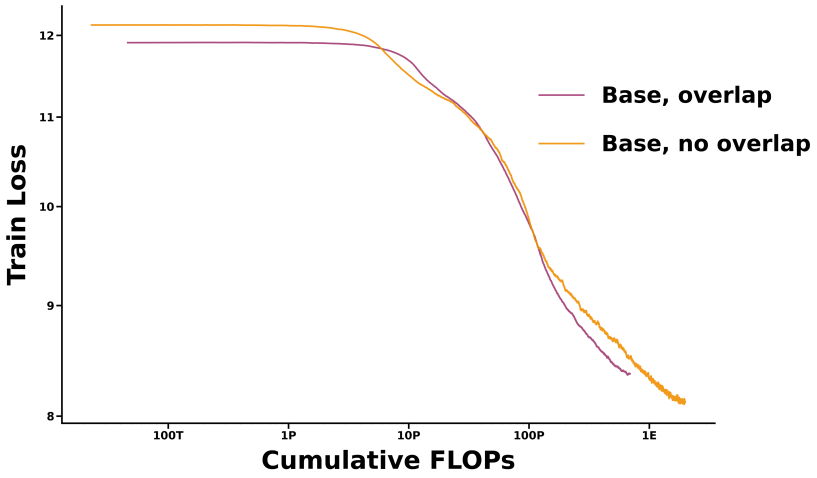


Figure 29: **Data Augmentation Comparison.** In overlap, episodes in the train dataset are converted into context for the model via a sliding-window approach. In no overlap, episodes are split into nearly non-intersecting context windows.

D VISUALIZATIONS

Task execution videos are available at: `https://osf.io/qmhrx/` or in the supplementary materials.

E CODE & DATA AVAILABILITY

Code, pretrained models, and datasets will be made publicly available after the peer review process.