LEARNING TO ACT

Session 3: From Research to Practice

Dana Kulić
Monash University
dana.kulic@monash.edu

RVSS 2025

Behavioural Cloning –problems revisited

- How to collect the expert data?
- What is the right state representation? Does the robot see the same things as the expert does?
- Expert demonstrations may cover only a very small region of the state-space
  - For large state/action spaces, may require a huge data collection effort
- What should the robot do when it encounters a situation that wasn't seen in the dataset?
- How to handle variations in strategy?

Today's Session

- Commercial efforts – bringing together perception and action
- How do we get the expert data?
  - Mechanisms for collecting demonstration data
  - Other types of expert feedback
- Revisiting RL
- RL+IL for the win?
- Outlook

π0: A Vision-Language-Action Flow Model for General Robot Control

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, Ury Zhilinsky

Physical Intelligence

https://www.physicalintelligence.company/blog/pi0

Motivation

- Large-scale models have been very effective at becoming generalists, but they are not *situated* on the physical world
- It would be great if robot policy could also be trained in a generalist way:
  - Train on highly diverse, large scale data
  - Fine-tune or prompt for specific task
- Challenges:
  - There needs to be enough diverse data for pre-training
  - What is the right model architecture?
  - What is the right training procedure?

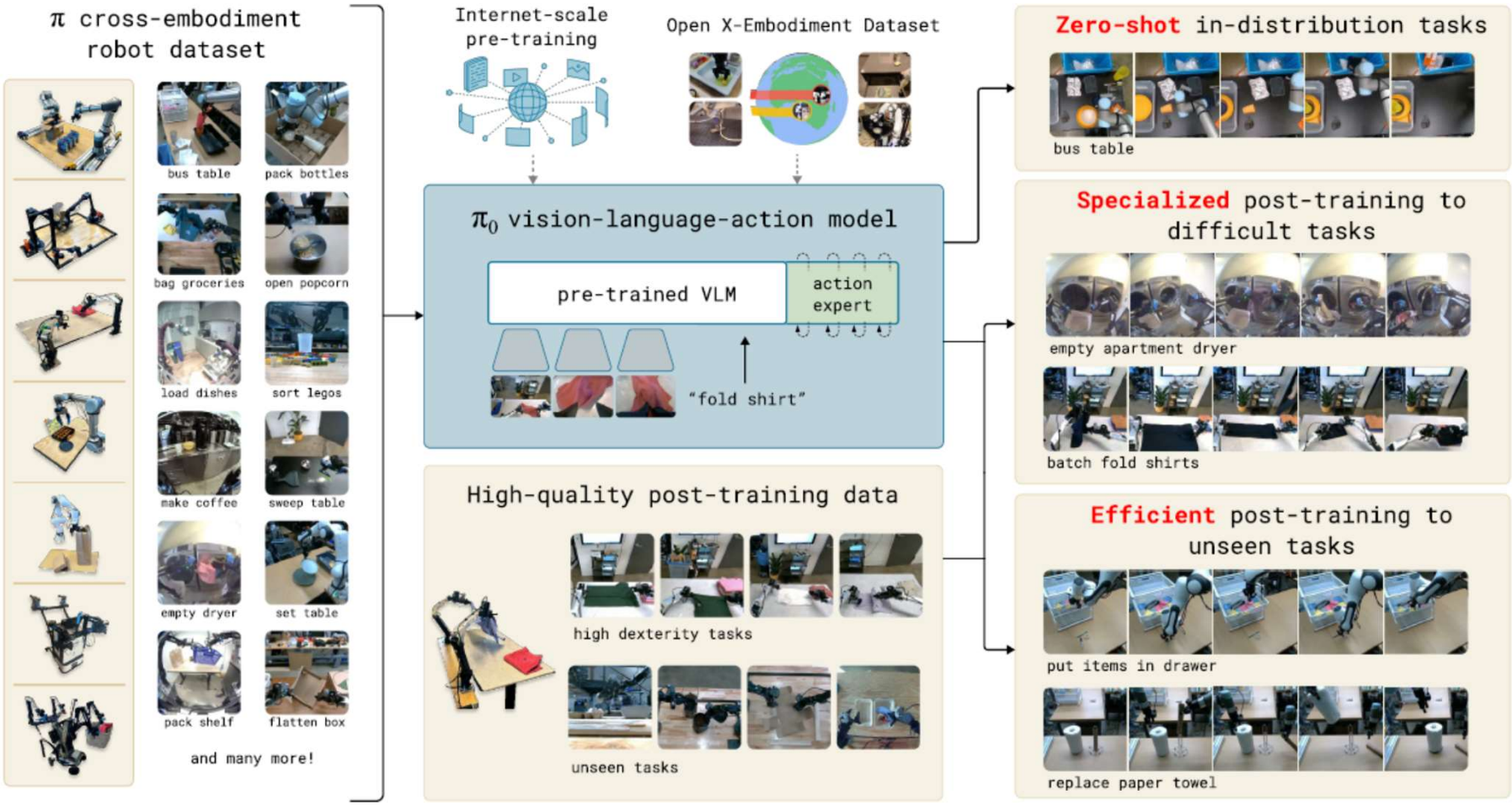


Fig. 1: Our generalist robot policy uses a pre-trained vision-language model (VLM) backbone, as well as a diverse cross-embodiment dataset with a variety of dexterous manipulation tasks. The model is adapted to robot control by adding a separate *action expert* that produces continuous actions via flow matching, enabling precise and fluent manipulation skills. The model can then be prompted for zero-shot control or fine-tuned on high-quality data to enable complex multi-stage tasks, such as folding multiple articles of laundry or assembling a box.

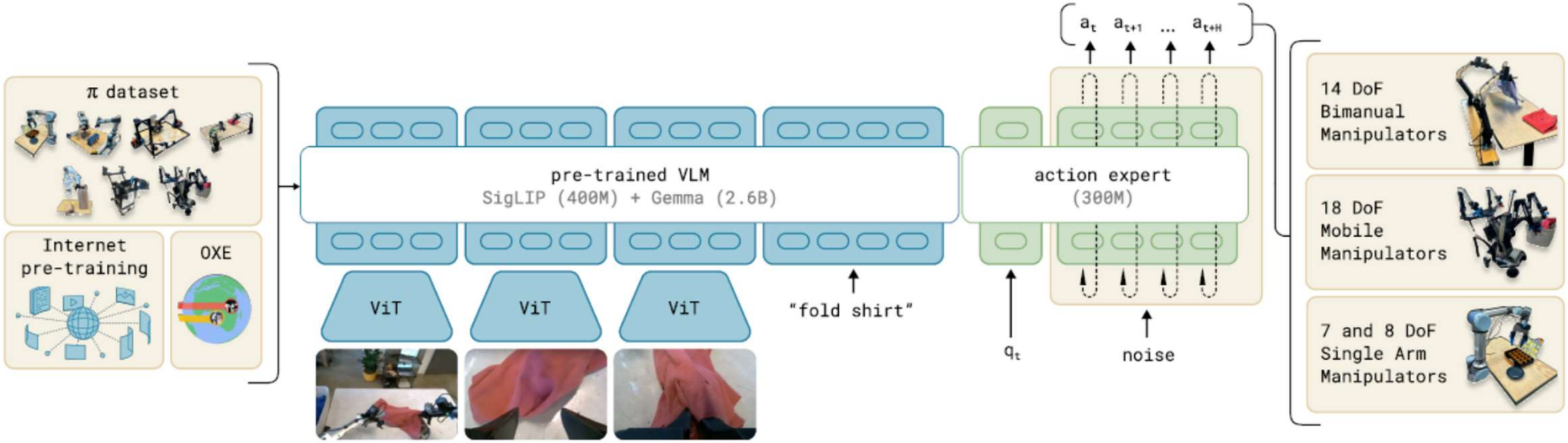


Fig. 3: **Overview of our framework.** We start with a pre-training mixture, which consists of both our own dexterous manipulation datasets and open-source data. We use this mixture to train our flow matching VLA model, which consists of a larger VLM backbone and a smaller *action expert* for processing robot states and actions. The VLM backbone weights are initialized from PaliGemma [5], providing representations learned from large-scale Internet pre-training. The resulting π_0 model can be used to control multiple robot embodiments with differing action spaces to accomplish a wide variety of tasks.

The model

- Pre-trained vision language model (PaliGemma)
  - Image encoders to embed images into the same embedding space as language tokens
  - Add robot-specific inputs (proprioceptive state) and outputs (robot actions)
- Flow matching to model the continuous distribution of actions
- Model $p(\mathbf{A}_t|\mathbf{o}_t)$,

$$\mathbf{A}_t = [\mathbf{a}_t, \mathbf{a}_{t+1}, ..., \mathbf{a}_{t+H-1}]$$

$$\mathbf{o}_t = [\mathbf{I}_t^1, ..., \mathbf{I}_t^n, \ell_t, \mathbf{q}_t],$$

- Action is trained in a diffusion like process

Data Collection and Training

- Pre-training: combination of open-source, private datasets with a large variety of robot embodiments and tasks
- Post-training: fine-tuning with smaller task-specific dataset

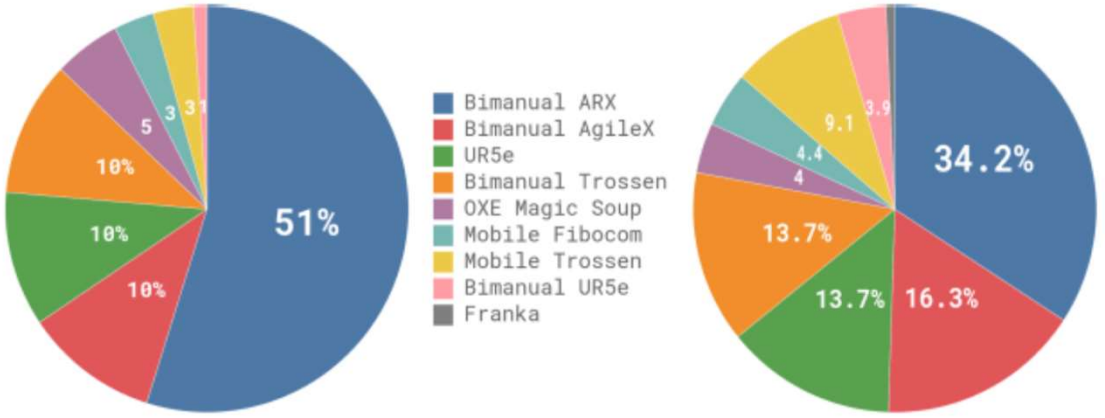


Fig. 4: **Overview of our dataset**: The pre-training mixture consists of a subset of OXE [10] and the π dataset. We use a subset of OXE, which we refer to as OXE Magic Soup [24]. The right figure illustrates the weight of the different datasets in the pre-training mixture. The left figure illustrates their relative sizes as measured by the number of steps.

Experiments

- How well does pi0 perform after pre-training on seen tasks?
- How well does pi0 follow language commands?
- How does pi0 compare to task-specific policies?
- Can pi0 be used for complex, multi-stage tasks?

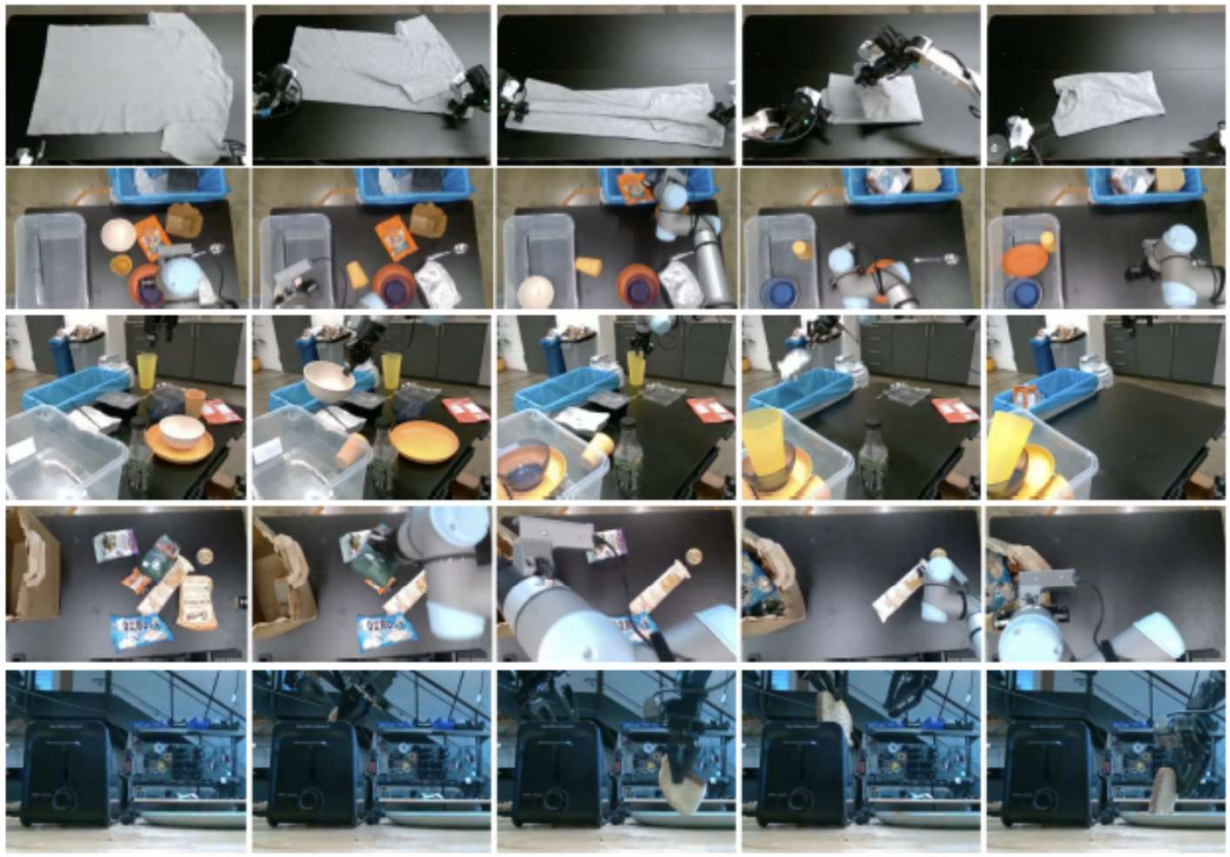


Fig. 6: **Zero-shot evaluation tasks**: To evaluate our base model, we run it after pre-training on five tasks: **shirt folding**, **bussing easy**, **bussing hard**, **grocery bagging**, and **toast out of toaster**. The tasks require a combination of dexterous manipulation, multi-stage behaviors, and semantic recognition.

https://www.physicalintelligence.company/blog/pi0

Results

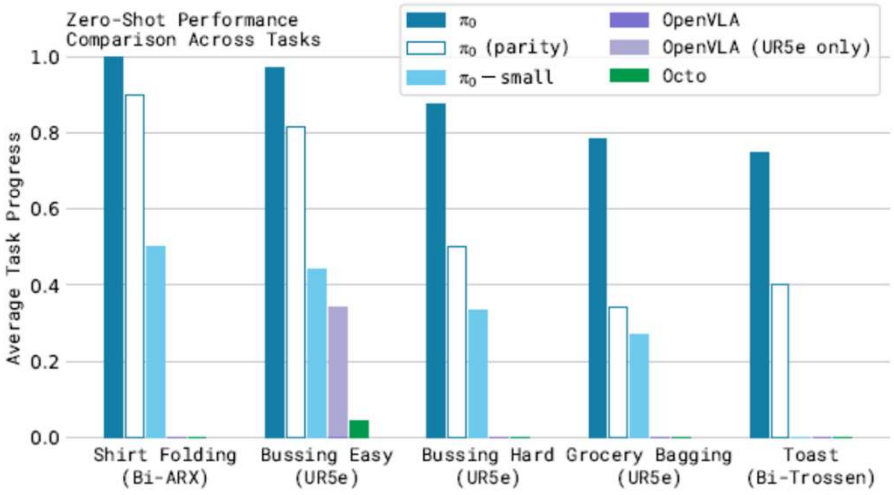


Fig. 7: **Zero-shot evaluation results**: We evaluate π_0 trained for the full 700k steps, a version trained for 160k steps that matches the number of updates for baseline models, π_0-small, and three baselines: OpenVLA and Octo trained on all of our data, and OpenVLA trained only on the UR5e tasks (which we found to work better on UR5e tasks). Across all tasks and all comparisons, even the "parity" version of our model outperforms all baselines, and the full version of our model achieves the best results by a large margin.

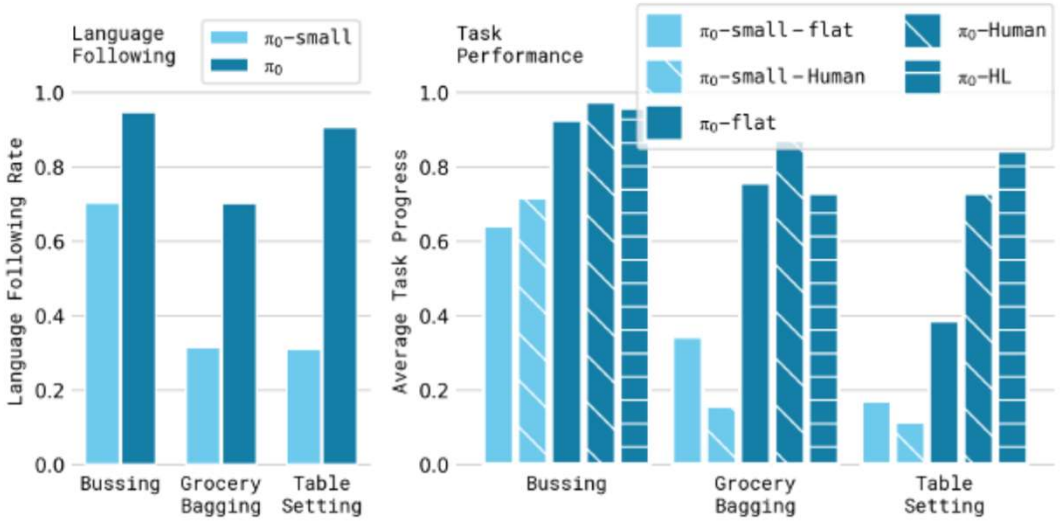


Fig. 9: **Language evaluation.** We compare "flat" versions of our policies, $-$`flat`, which receive only the overall task command (e.g., "bag the groceries") with a method that receives intermediate commands from a human expert, $-$`human`, or a high-level VLM policy, $-$`HL`. We also compare our model to a small non-VLM variant under the "expert" condition, π_0 and π_0-small, in terms of language following accuracy. The results show a significant improvement with π_0 from intermediate language commands provided by a human expert and to a lesser degree by an autonomous high-level policy. Notably, due to π_0-small's limited language following ability, overall it does not gain with the addition of a high-level expert.

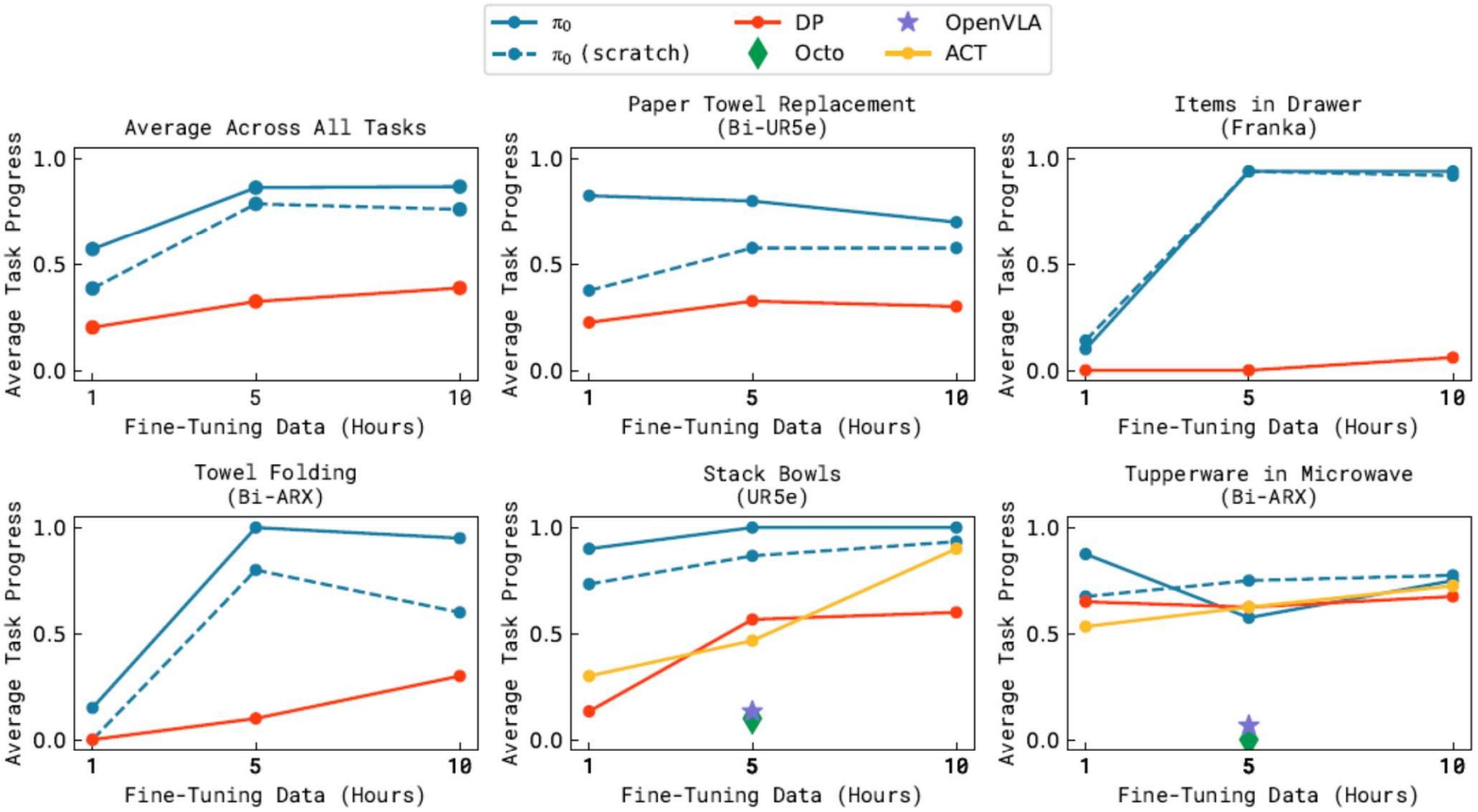


Fig. 11: **Fine-tuning with varying amounts of data.** π_0 can learn some easier tasks even with smaller amounts of data, and the pre-trained model often attains a larger improvement over the model trained from scratch.

Complex Tasks

- Laundry folding
- Mobile laundry
- Dryer unloading
- Table bussing
- Box-building
- To-go box
- Packing eggs

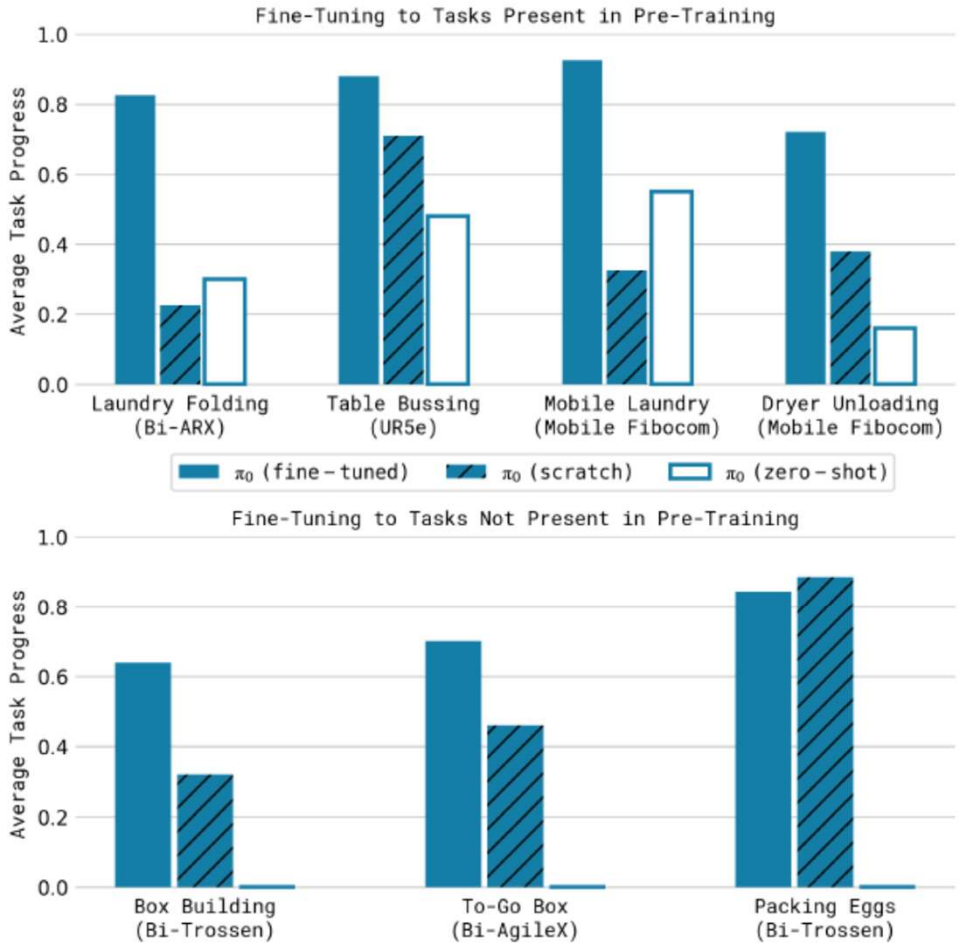


Fig. 13: **Post-training results on complex tasks** in terms of average scores over 10 trials. The full pre-trained π_0 model attains more than 50% of the maximum score across all of the tasks, and typically outperforms the ablations, with especially significant improvements on the hardest tasks.

How to collect expert data?

From human demonstrations directly

Via teleoperation

How to collect demonstration data?

Properties of demonstration data

- Do I get the state only or both the state and action?
- How valid is the assumption that the expert data is optimal?
  - Intuitive interface
  - Sensor noise
  - Demonstrator expertise
- How much of the state-space/task variation is covered?
- Are there variations in strategy?

Are there other ways to collect expert input?

Are there other ways to collect expert input?

- Preference Learning:
  - Use preferences to learn the human reward function - Reinforcement Learning from Human Feedback (RLHF, Christiano et al. 2017)
  - Use preferences to learn the policy directly – Direct Preference Optimisation (DPO – Rafailov et al. 2023)
  - Make the process more efficient by encoding prior on likely human biases – Kahneman-Tversky Optimisation (KTO – Ethayarajh et al. 2024)
- Unifying multiple human signals for learning (Jeon et al., 2020)

Revisiting RL

Legged Locomotion in Challenging Terrains using Egocentric Vision

A. Agarwal, A. Kumar, J. Malik and D. Pathak, CoRL 2022

https://vision-locomotion.github.io/

Motivation and Approach

- How should visual information be used for walking? Do we need a detailed elevation map?
- Animals (and robots) can walk well without vision on easy terrain
- Vision is useful for walking over challenging terrain
- Humans look ahead at ground a few steps away – remembering the recently seen information for the current step

- Apply these design principles for robot walking:
  - build a short term memory of recent vision and action history
  - Train optimal policy using RL in simulation
- Implement on A1 robot, with all compute and sensing on board

Method

- Action space: joint angles of the robot, at 50Hz
- State space: proprioception and depth
- Two stage training approach:
  - Stage 1: RL using low res scandots as proxy for depth
  - Stage 2: supervised learning for full depth input
- Two architectures:
  - Rapid Motor Adaptation (RMA): separate heads for estimating latent vectors for local terrain geometry, environment parameters and proprioception
  - Monolithic: single RNN that maps from raw proprioception and vision data directly to joint angles

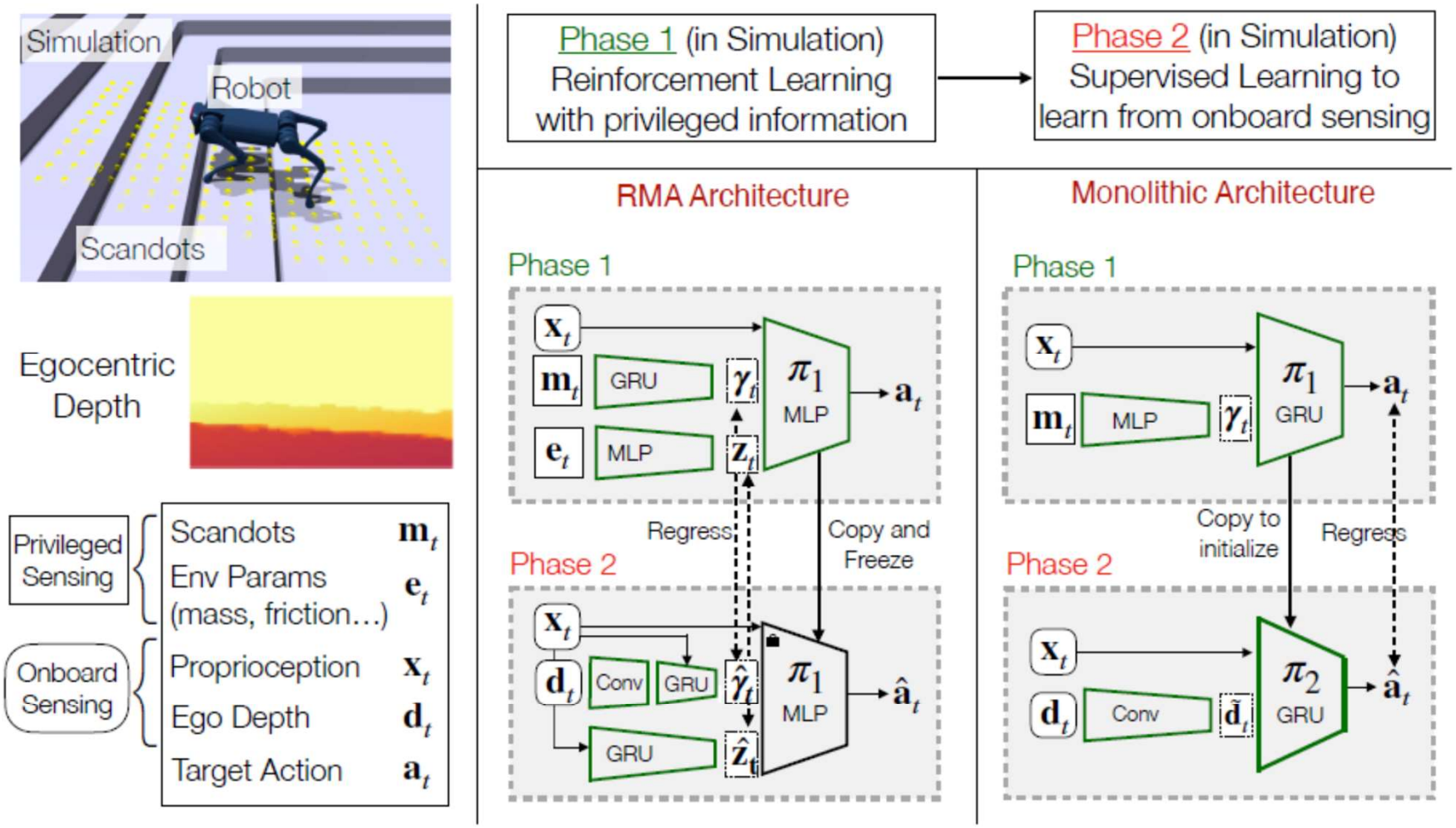


Figure 3: We train our locomotion policy in two phases to avoid rendering depth for too many samples. In phase 1, we use RL to train a policy π^1 that has access to scandots that are cheap to compute. In phase 2, we use π^1 to provide ground truth actions which another policy π^2 is trained to imitate. This student has access to depth map from the front camera. We consider two architectures (1) a monolithic one which is a GRU trained to output joint angles with raw observations as input (2) a decoupled architecture trained using RMA [3] that is trained to estimate vision and proprioception latents that condition a base feedforward walking policy.

Rewards

- *Absolute work penalty* $-|\boldsymbol{\tau} \cdot \mathbf{q}|$ where $\boldsymbol{\tau}$ are the joint torques. We use the absolute value so that the policy does not learn to get positive reward by exploiting inaccuracies in contact simulation.
- *Command tracking* $v_x^{\text{cmd}} - \left|v_x^{\text{cmd}} - v_x\right| - \left|\omega_z^{\text{cmd}} - \omega_z\right|$ where v_x is velocity of robot in forward direction and ω_z is yaw angular velocity (x, z are coordinate axes fixed to the robot).
- *Foot jerk penalty* $\sum_{i \in \mathcal{F}} \|\mathbf{f}_t^i - \mathbf{f}_{t-1}^i\|$ where $\mathbf{f}_t^i$ is the force at time t on the i^{th} rigid body and $\mathcal{F}$ is the set of feet indices. This prevents large motor backlash.
- *Feet drag penalty* $\sum_{i \in \mathcal{F}} \mathbb{I}\left[f_z^i \geq 1\text{N}\right] \cdot \left(\left|v_x^i\right| + \left|v_y^i\right|\right)$ where $\mathbb{I}$ is the indicator function, and v_x^i, v_y^i is velocity of i^{th} rigid body. This penalizes velocity of feet in the horizontal plane if in contact with the ground preventing feet dragging on the ground which can damage them.
- *Collision penalty* $\sum_{i \in \mathcal{C} \cup \mathcal{T}} \mathbb{I}\left[\mathbf{f}^i \geq 0.1\text{N}\right]$ where $\mathcal{C}, \mathcal{T}$ are the set of calf and thigh indices. This penalizes contacts at the thighs and calves of the robot which would otherwise graze against edges of stairs and discrete obstacles.
- *Survival bonus* constant value 1 at each time step to prioritize survival over following commands in challenging situations.

Phase 1 Training

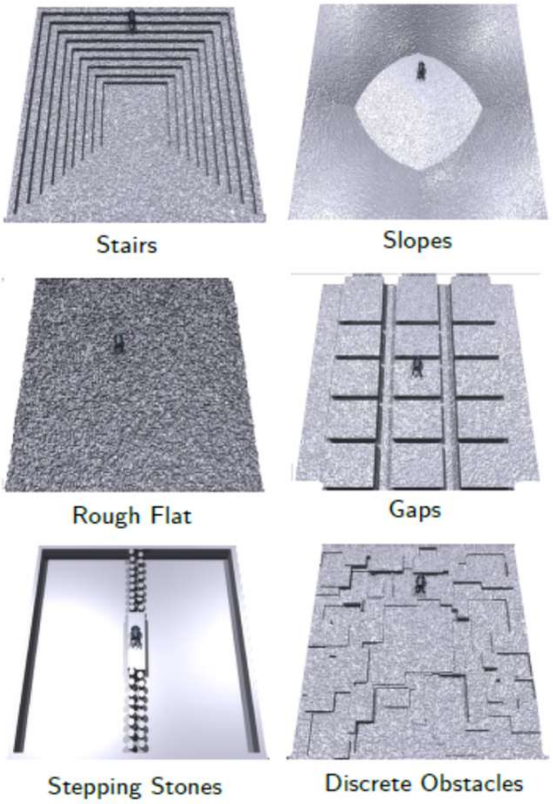


Figure 5: Set of terrain we use during training

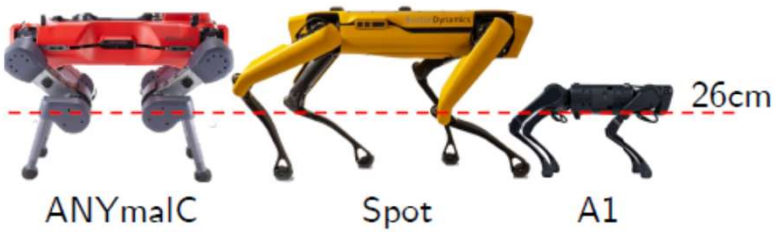
Name	Range
Height map update frequency*	[80ms, 120ms]
Height map update latency*	[10ms, 30ms]
Added mass	[−2kg, 6kg]
Change in position of COM	[−0.15m, 0.15m]
Random pushes	Every 15s at 0.3m/s
Friction coefficient	[0.3, 1.25]
Height of fractal terrain	[0.02m, 0.04m]
Motor Strength	[90%, 110%]
PD controller stiffness	[35, 45]
PD controller damping	[0.4, 0.6]

Table 3: Parameter randomization in simulation. * indicates that randomization is increased to this value over a curriculum.

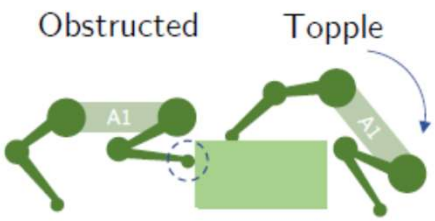
Phase 2: Supervised learning

- Use phase 1 policy as labels to train policy that only has access to on-board data
- For RMA architecture, only train the feature extractors

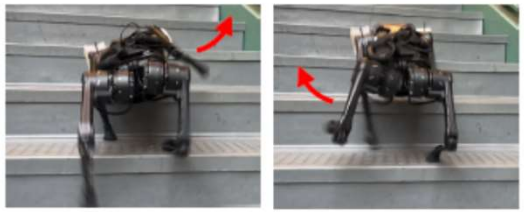
Experiments



(a) Robot size comparison



(b) Challenges due to size



(c) Emergent hip abduction

Figure 2: A smaller robot (a) faces challenges in climbing stairs and curbs due to the stair obstructing its feet while going up and a tendency to topple over when coming down (b). Our robot deals with this by climbing using a large hip abduction that automatically emerges during training (c).

- Adopt policy trained in simulation zero-shot
- Compare to baselines:
  - Blind: no vision, only proprioception
  - Noisy elevation maps

Results – Physical Experiments

Upstairs
17cm high, 30cm deep

	Success	#Stairs
Ours	100%	13
Blind	0%	2.2

Downstairs
17cm high, 30cm deep

	Success	#Stairs
Ours	100%	13
Blind	100%	13

Stepping Stones
30cm wide, 15cm apart

	Success	#Stones
Ours	94%	9.4
Blind	0%	0

Gaps
26cm apart

	Success
Ours	100%
Blind	0%

Figure 4: We show success rates and time-to-failure (TTF) for our method and the blind baseline on curbs, stairs, stepping stones and gaps. We use a separate policy for stairs which is distilled to front camera, and use a separate policy trained on stepping stones distilled to the top camera which we use for gaps and stepping stones. We observe that our method solves all the tasks perfectly except for the stepping stone task in which the robot achieves 94% success. The blind baseline fails completely on gaps and stepping stones. For upstairs, it makes some progress, but fails to complete the entire staircase even once, which is expected given the small size of the robot. The blind policy completes the downstairs task 100% success, although it learns a very high impact falling gait to solve the task. In our experiments, the robot dislocates its real right leg during the blind downstairs trials.

Decision Transformer: Reinforcement Learning via Sequence Modeling

L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, I. Mordatch

https://sites.google.com/berkeley.edu/decision-transformer

https://arxiv.org/abs/2106.01345

Motivation and Proposed Approach

- Existing DRL algorithms use interaction data to approximate the policy or the value function using NNs (NNs are used for function approximation)
- Recent research has shown the potential of Transformer architecture for generative (predictive) sequence modelling

- This paper: recast RL as a sequence modelling problem
  - Train a Transformer to model the joint distribution of states, actions and returns-to-go
  - Prompt with a high reward, use the learned model to predict the action

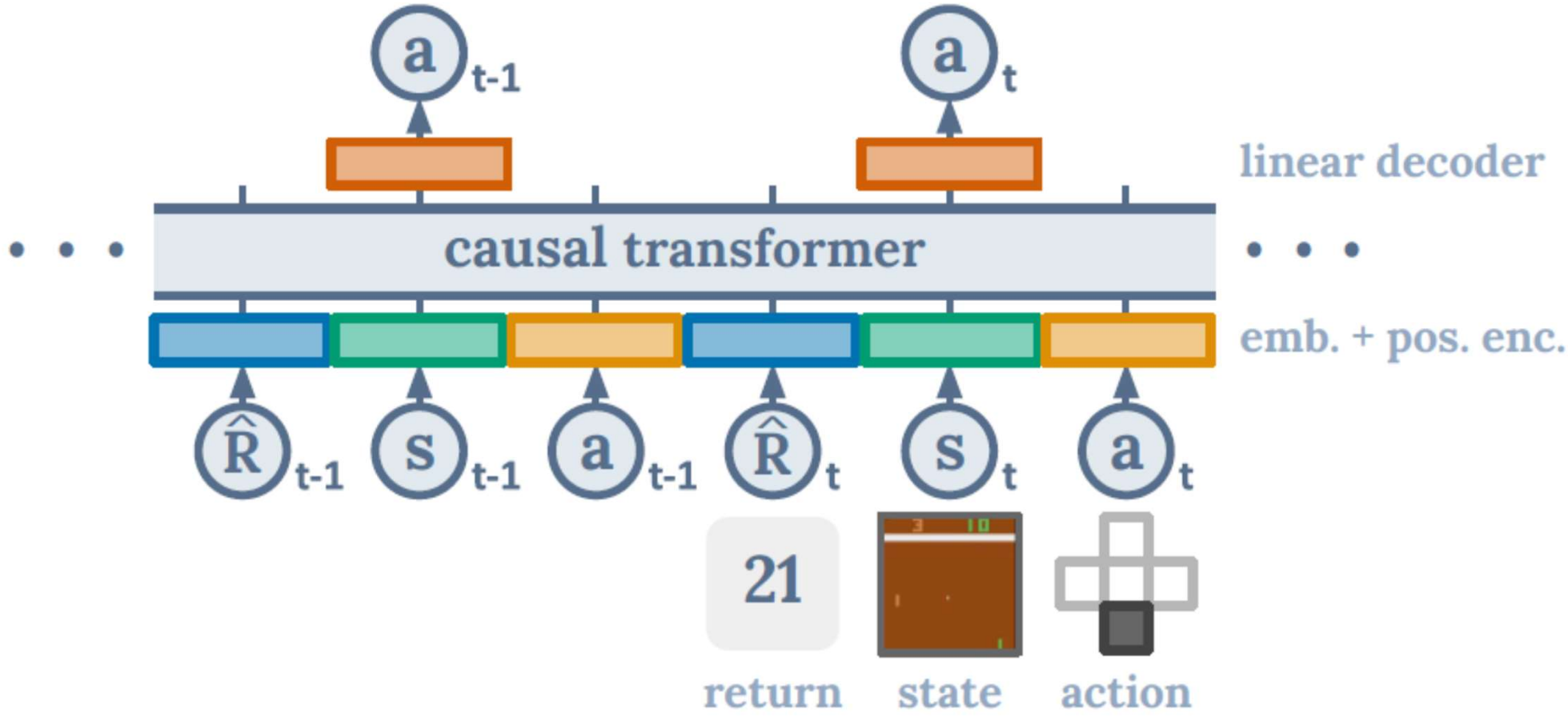


Figure 1: Decision Transformer architecture[1]. States, actions, and returns are fed into modality-specific linear embeddings and a positional episodic timestep encoding is added. Tokens are fed into a GPT architecture which predicts actions autoregressively using a causal self-attention mask.

Method

- Represent the reward in the trajectory as the return-to-go $\widehat{R}_t = \sum_{t'=t}^{T} r_{t'}$

$$\tau = \left(\widehat{R}_1, s_1, a_1, \widehat{R}_2, s_2, a_2, \ldots, \widehat{R}_T, s_T, a_T\right)$$

- At test time, specify the initial state and the desired performance as the conditioning information, predict the next action
- Learn tokens for each modality, together with embedding for each time step
- Given a dataset of offline trajectories, train to predict actions

Evaluation

- Multiple simulation testbeds:
  - Atari:
    - High dimensional input (images)
    - Difficulty of credit assignment
    - Discrete actions
  - OpenAI Gym
    - Continuous control tasks
- Training data is from a learner in various stages (i.e., not optimal policy execution)
- Baselines:
  - TD learning
  - Imitation learning

Results - Summary

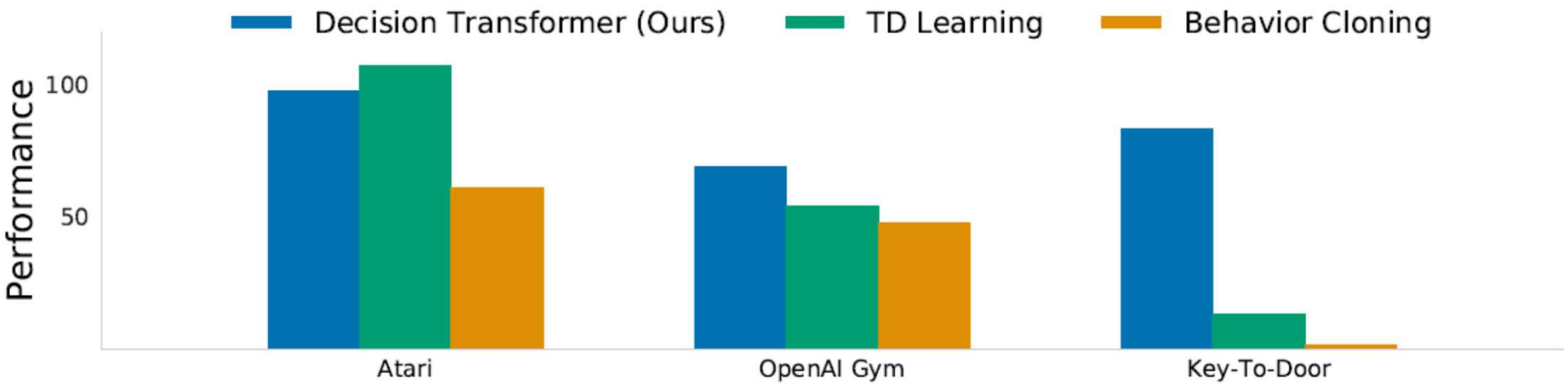


Figure 3: Results comparing Decision Transformer (ours) to TD learning (CQL) and behavior cloning across Atari, OpenAI Gym, and Minigrid. On a diverse set of tasks, Decision Transformer performs comparably or better than traditional approaches. Performance is measured by normalized episode return (see text for details).

Results – OpenAI Gym

Dataset	Environment	DT (Ours)	CQL	BEAR	BRAC-v	AWR	BC
Medium-Expert	HalfCheetah	**86.8 ± 1.3**	62.4	53.4	41.9	52.7	59.9
Medium-Expert	Hopper	107.6 ± 1.8	**111.0**	96.3	0.8	27.1	79.6
Medium-Expert	Walker	**108.1 ± 0.2**	98.7	40.1	81.6	53.8	36.6
Medium-Expert	Reacher	**89.1 ± 1.3**	30.6	-	-	-	73.3
Medium	HalfCheetah	42.6 ± 0.1	44.4	41.7	**46.3**	37.4	43.1
Medium	Hopper	**67.6 ± 1.0**	58.0	52.1	31.1	35.9	63.9
Medium	Walker	74.0 ± 1.4	79.2	59.1	**81.1**	17.4	77.3
Medium	Reacher	**51.2 ± 3.4**	26.0	-	-	-	**48.9**
Medium-Replay	HalfCheetah	36.6 ± 0.8	46.2	38.6	**47.7**	40.3	4.3
Medium-Replay	Hopper	**82.7 ± 7.0**	48.6	33.7	0.6	28.4	27.6
Medium-Replay	Walker	**66.6 ± 3.0**	26.7	19.2	0.9	15.5	36.9
Medium-Replay	Reacher	**18.0 ± 2.4**	**19.0**	-	-	-	5.4
Average (Without Reacher)		**74.7**	63.9	48.2	36.9	34.3	46.4
Average (All Settings)		**69.2**	54.2	-	-	-	47.7

Table 2: Results for D4RL datasets[3]. We report the mean and variance for three seeds. Decision Transformer (DT) outperforms conventional RL algorithms on almost all tasks.

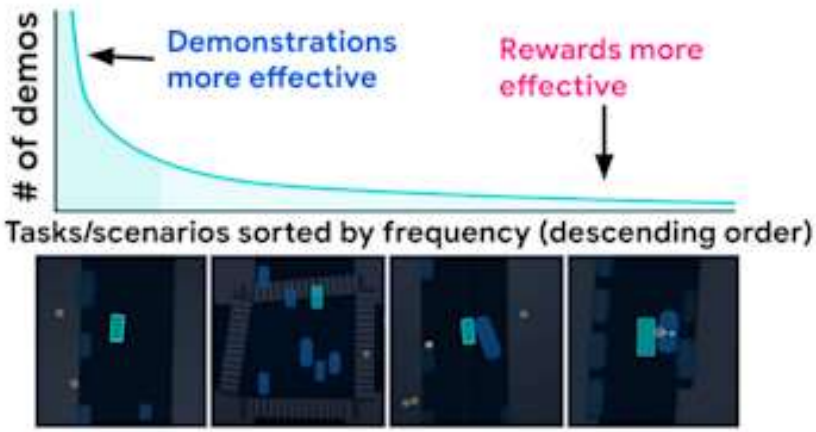
What is DT learning?

- It’s not just doing behavioural cloning on a well selected subset of the data
- DT seems to model the distribution of returns
- Context length is important – longer is better
- Assuming context length is appropriate, DT can perform effective long-term credit assignment
- DT can also be used to predict the value (critic)
- DT is robust to sparse rewards

Combining RL + IL

- Inverse Reinforcement Learning
  - Use the demonstrations (or other user inputs) to infer the reward, then train policy to optimise inferred reward
- RL/Optimisation as a data generator for IL (e.g., Clever et al., 2017)
- Robustify Imitation with RL (e.g., Lu et al., 2022)

BC-SAC (Lu et al., 2022)



Critic Objective

$$\min_Q \ \mathbb{E}_{s,a,s' \sim \pi}[(Q(s,a) - \hat{Q}(s,a,s'))]^2,$$

where

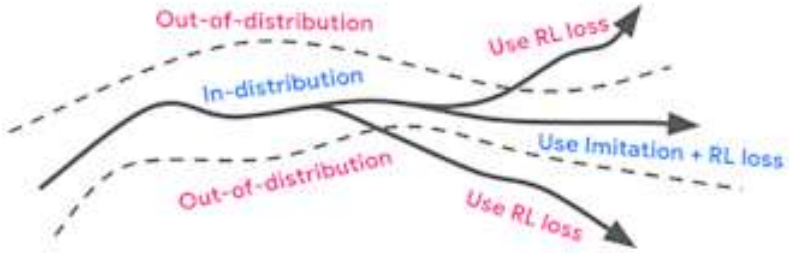
$$\hat{Q}(s,a,s') = r(s,a) + \gamma \mathbb{E}_{a' \sim \pi}\left[\bar{Q}(s',a') - \log \pi(a'|s')\right]$$

Actor Objective

$$\mathbb{E}_{s,a \sim \pi}[Q(s,a) + \mathcal{H}(\pi(\cdot|s))] + \lambda \mathbb{E}_{s,a \sim \mathcal{D}}[\log \pi(a|s)]$$

Reward

$$R = R_{\text{collision}} + R_{\text{off-road}}$$



https://waymo.com/research/imitation-is-not-enough-robustifying-imitation-with-reinforcement-learning/

Summary and Outlook

Open Questions:

- Better and less effortful ways of collecting demonstration/expert data
- More efficient use of the data we have
  - Representation for action
- More efficient use of prior knowledge
  - Physics
  - Context
- On-line, continuous learning?
  - From collaborative execution?