CENTRILLION TECHNOLOGIES

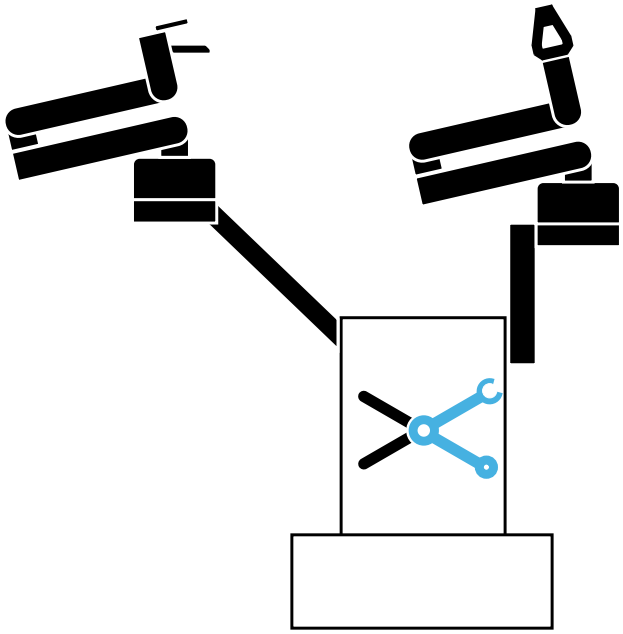
ALOHA Mobile Robot

Training Parameters

Model Training Options:

1. LAMBDA Lab:
   - 80 GB Nvidia H100 with 16,896 CUDA Cores
2. System76:
   - 8 GB GeForce RTX 4060 with 3,072 CUDA Cores
3. Omen Desktop:
   - 24 GB Nvidia Titan RTX with 4,608 CUDA Cores
   - 12 GB Nvidia GeForce with 7,680 CUDA Cores
4. In-House Server:
   - 24 GB Nvidia GeForce RTX 4,090 with 16384 CUDA Cores
5. Google Colab:
   - 40 GB Nvidia A100 with 3,456 FP64 CUDA Cores

Training Pipeline

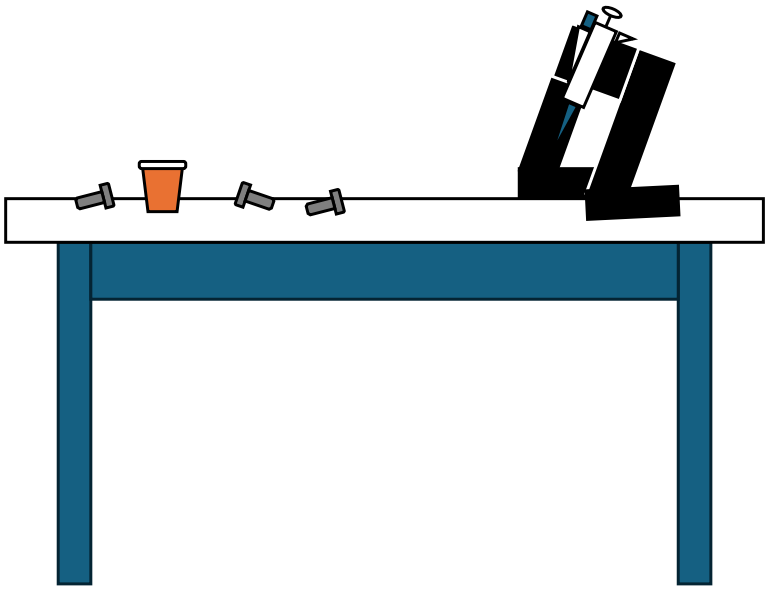


Episode Collection:

Using the back set of “Leader” arms, an operator puppets the front manipulators to complete a task. The data generated by this process is compressed and stored in .hdf5 files.

- **Qpos**
  - Encoder data representing the angle of each motor in the robotic arm
- **Effort**
  - The power required to take the joint from the current to desired position
- **Image**
  - 480 x 848 px images from each camera
- **Depth**
  - Integer depth value derived from 480 x 848 depth map
- **Base Action:**
  - Motor data for the Slate base of the ALOHA platform.

Training Pipeline



Verification:

Utilizing the .ckpt and .pkl files produced in the model training, verification occurs. This largely consists of repeated testing of the ALOHA's ability to autonomously complete the given task, with some stress testing to identify the limits of the program

- **Normalization**
  - Do small changes to the environment change the robot's behaviors?
- **Range**
  - What is the area in which a task can be completed?
- **Accuracy**
  - How well does the robot complete the task?
- **Repeatability**
  - Does the system, under identical conditions, complete the task the same way?

Training Pipeline

```
(aloha_venv) aloha@aloha-pc: ~/aloha_training/act_plus_plus$
python3 imitate_episodes.py --task_name aloha_mobile_syringe --
ckpt_dir ~/aloha_data/aloha_mobile_syringe/ckpt_0806_4090_long -
-policy_class ACT --kl_weight 10 --chunk_size 100 --hidden_dim 512 --
batch_size 18 --dim_feedforward 3200  --num_steps 35000 --lr 1e-5 --
seed 0 --eval_every 1000000000 --torque_base --save_every 1000 –
eval
```

Model Training:

Utilizes a modified branch of the Trossen code for imitation learning and the .hdf5 files, or episodes, collected earlier. This step is where hyperparameters are tuned to improve the model as much as possible. This stage also sees the selection of the model used in training

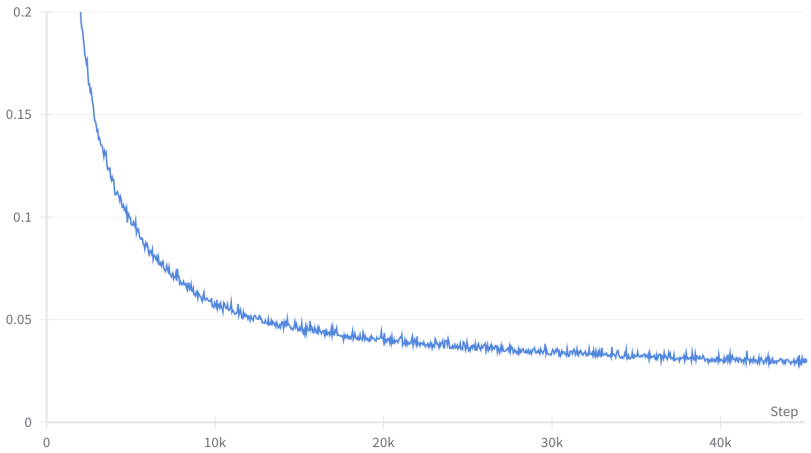
- **Num Steps**
  - Number of epochs the system iterates through to produce the model
- **Batch Size**
  - Number of samples processed together in one training iteration
- **Chunk Size**
  - Determines the number of time-steps a policy determines before collecting new image and qpos data.
- **Learning Rate**
  - Learning rate for the optimizer

Policy Performance

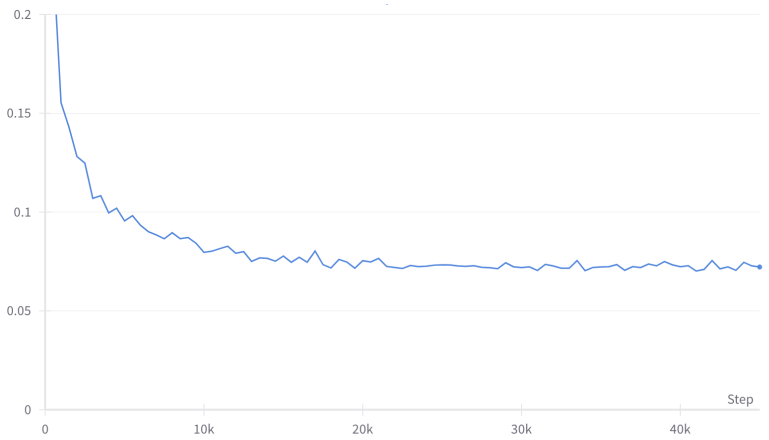
Moving Wafer

`Batch size : 62  | Chunk size: 150  | # of iteration: 50,000 | ACT algorithm`

Loss



Validation Loss



Models and Policies

- Imitation Learning (IL)
- ACT Policy
- Incorporate IL with LLM

MDP:

The environment has state space, *S*; action space, *A*; a stochastic transitional model *P* which is dominated by policy π, i.e. $P_\pi(s_{t+1}|s,a)$, $a = \pi(s)$. Each motion is an episode (trajectory), modeled as $\tau = (s_0, a_0, s_1, a_1, \dots)$.

Pipeline:

1. Collect embodied trajectories (episodes) of state-action pairs, $\tau^* = (s_0, a_0, s_1, a_1, \dots)$, dominated by a latent expert policy π^*.
2. Use an AI model (MLP, LSTM, transformer etc.) as policy π, train it as close to π^* as possible; in deployment, use the trained policy to control the robot

Application Range:

1. The motion can be modeled as Markov Decision Process (MDP)
2. The motion can be done well by human, but hard for robot with traditional controllers

In Practice

Methods:

1. Collect demonstrations (τ^* trajectories) from expert
2. Treat the demonstrations as iid state-action pairs: $(s_0^*, a_0^*), (s_1^*, a_1^*)$, ...
3. Learn π_θ policy using supervised learning by minimizing the loss function $L(a^*, \pi_\theta(s))$

Our State and Action

- State: 14-robot arms motor joint positions; 2-base robot joint positions; 3-camera data (each frame 480*848*3)
- Action: 14+2 joint positions (control the robot arm & base robot)

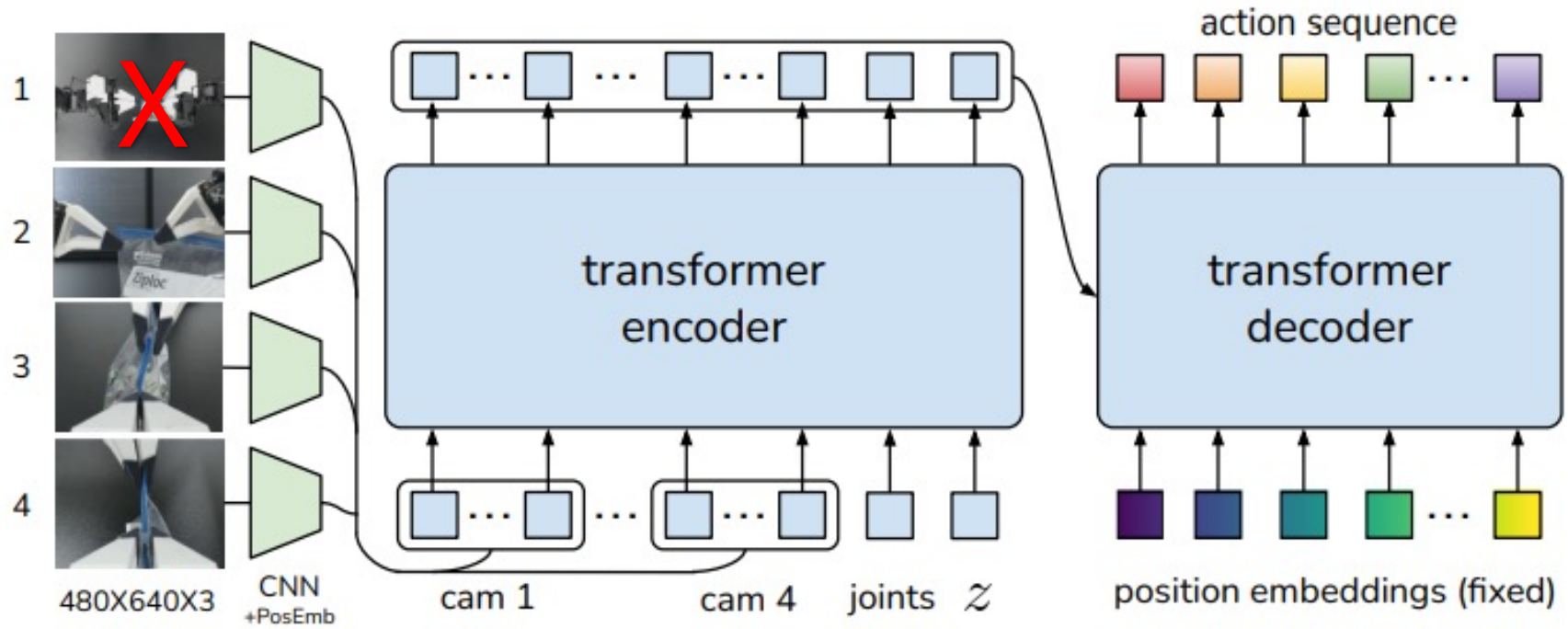
Action Chunking with Transformers (ACT) [1]

First things first, what is action chunking?

- In short, ACT policy packs up a sequence of actions as a chunk, rather than a single action, i.e. $\pi(a_{t:t+k}|s_t)$ rather than $\pi(a_t|s_t)$, where k is the chunk size.

[1]. Zhao, T. Z., Kumar, V., Levine, S., & Finn, C. (2023). Learning fine-grained bimanual manipulation with low-cost hardware. arXiv preprint arXiv:2304.13705.

ACT Training Phase: Generate Actions



Conditional Variational Autoencoder (CVAE) structure (encoder-decoder), we are training transformer encoder, decoder's attention weights.

Imitation Learning Meets LLM

- Imitation learning can conduct a single motion of around 20s
- What if we have a sequence of motions that we want our robot to do? If using IL for long sequence (> 1 min):
  - Very hard to train the long episode
  - No DoF for different sequence. For instance, {act1, act2, act3} and {act3, act1, act2} are different episodes.
- Luckily, we have LLM these days. A very good tool for simple reasoning and task planning

Methods

1. We have a library of different single motions conduct by IL
2. We first prompt LLM with the names of the actions in the library. Letting it know what it can do.
3. Then we prompt LLM with our long motion instruction, letting it know what we want it to do.
4. LLM would output a structured list of motions that fulfill our instructions.
5. After our confirmation, robot would run the sequence of tasks

Preliminary Test:

```
(aloha_venv) aloha@aloha-pc:~/aloha_training/act_plus_plus$ python llm_robot.py
I am a bio-science experiment helper robot working at Centrillion Tech., what you want me to do today?

Enter the task instruction please:you are at the table, collect the wafer, then come back

LLM ANSWER:
Sure, let's break down the task instruction and create a sequence of simple motions:

1. Analyze the Task:
   - The starting position is at the table.
   - The first action is to collect the wafer.
   - The final action is to move back to home.

2. Create the sequence of simple motions:
   - Since you are already at the table, skip "move forward to table".
   - Collect the wafer.
   - Move back to home.

Here is the sequence of simple motions:

['collect the wafer', 'move back to home']

In summary, I will conduct the following movement in sequence:  ['collect the wafer', 'move back to home']

Press Enter to continue, or any other key to exit.
```

Training Results

- Pick syringe and push it down
- Pick up screw(s)
- Move wafer
- Incorporate IL with LLM

Results

Pick up and push syringe

Results

Pick up screw

Results

Moving Wafer

Results

Moving Wafer (in different position)

Results

Moving Wafer (without the box)

Results

LLM (Collecting Wafer + Moving back home)