

Multimodality - why regression policies average and Claude does not

Notes from a chat with Claude, October 3–4, 2026.

Question: Explain the multimodality problem in more depth. If I send the camera pictures to Claude and ask how to walk without hitting an obstacle, it answers correctly, so why would a robot policy not? What am I missing? Explain from first principles.

You're not misunderstanding the problem; your intuition is actually the clue. Claude doesn't average because it isn't built to output one best number. It outputs a probability for every possible next word and then picks one. An old-style robot policy outputs one number trained to minimize squared error, and the number that minimizes squared error is the average. Modern policies fixed this by copying Claude's trick. Separately, "Claude giving advice" and "a policy driving motors" differ in other important ways, covered below.

1. Why a regression policy picks the crash

Take one camera image of the obstacle. Half the demonstrators steered 30° left and half steered 30° right. The policy outputs a single steering angle, and training adjusts it to make the average squared error as small as possible. Score three candidate answers:

Policy outputs	Error vs. left demos	Error vs. right demos	Average loss
30° left	0	$60^2 = 3,600$	1,800
Straight (0°)	$30^2 = 900$	900	900, the lowest
30° right	3,600	0	1,800

By the training objective's own scoring, driving straight into the obstacle is the best answer.

The general reason is that squared error splits into two parts:

average loss = (spread of the demos) + (distance from your guess to their mean)²

You can't change the spread, so the loss is smallest when you guess the mean.

Two consequences:

- **It isn't a vision failure.** The network can see the obstacle perfectly. The training target itself says "the best single guess is the middle."
- **Averaging only hurts when the valid options are separated.** If anything between 10° and 20° works, the average is also valid. Here the valid set is left *or* right with a gap in between. Robotics is full of these either/or situations: which side to pass on, which handle to grab, which sleeve first, which hand to use, wait or go.

2. Why Claude doesn't do this

- **It outputs a distribution, not a value.** For each next word, Claude assigns a probability to every possible word. Its training rewards putting probability on what actually came next. If the data says "left" half the time and "right" half the time, it learns 50% and 50%, keeping both options instead of blending them.
- **It commits, then stays consistent.** It picks one word, say "left", and every following word is conditioned on that choice, so the plan is coherent.
- **There is nothing in between.** No word sits halfway between "left" and "right", so averaging isn't even possible.

Ask Claude ten times and some answers will say left and some right, but none will say "straight through." That is multimodality handled correctly: each answer is one good option, never a blend of two.

The same thing happens in images. Train an image model with squared error and you get blurry pictures, because the average of many possible sharp images is a blur. Diffusion models produce sharp images by sampling one possibility.

3. How robot policies copy the trick

- **Turn actions into tokens.** Cut each joint's range into 256 bins, predict a probability per bin, and pick one, just as Claude picks words. This is how RT-2 and OpenVLA work, and π 0-FAST compresses whole action sequences into tokens.
- **Diffusion or flow matching** (Diffusion Policy, π 0, GR00T). Start from random noise and let the network push it, step by step, toward something that looks like a real demonstrated action. Picture a landscape with two valleys, left and right: drop a ball anywhere and it rolls into one of them. The ridge between them is not a resting place, because no demos lie there.
- **A hidden "style" variable.** ACT gives the model a latent variable that captures which choice the demonstrator made. Fixing it at run time commits the policy to one style.
- **Commit over time.** A policy that re-picks every 20 ms could choose left now and right a moment later, wobbling between them. Predicting a one-second chunk at once commits to one plan, the same way Claude conditions each word on the previous ones.

4. What else separates Claude's advice from a policy

- **A plan is not an action.** "Go around on the left" is a plan. The policy must output the exact command for every motor 50 times a second (how far to bend each knee, where to put each foot) while the world moves.
- **Speed.** Claude takes seconds to answer, and the robot needs a decision every 20 ms. Waiting for the answer, it would already have hit the obstacle or fallen.
- **Precision.** Image models like Claude are good at *what* and roughly *where*. They are weak at exact distances, the robot's own body size, and contact, such as whether the hip clears the box by 3 cm.
- **Prior knowledge.** Claude learned from the internet. A policy trained from scratch on 200 demos knows only those demos.
- **Feedback.** The policy must keep re-deciding, and its own small errors change what it sees next.

5. Real systems use both

- **Inside one model:** a VLA (vision-language-action model) is a Claude-like vision-language model plus a generative action head. The language half supplies the understanding ("that's an obstacle, go around it"). The action head turns it into precise motor commands without averaging.
- **As two layers:** a slow, Claude-like model chooses what to do, and a fast policy does it. Figure's Helix plans at 7–9 Hz and controls at 200 Hz; GR00T N1 runs its action head at 120 Hz.

What you were missing is narrow. Your instinct was right that the knowledge to pick a side lives in large pretrained models, which is why VLAs are built on them. But the crash comes from the output format and the training objective, not from a lack of understanding. And Claude's correct answer is a plan, not the fast, precise stream of motor commands a robot needs.

A quick check to test it on a new case: sometimes the demonstrator pauses one second before grasping, and sometimes grabs immediately. What would a squared-error policy do at that moment, and what would a flow-matching policy do?