

PAPER REVIEW

A Vision-Language-Action Model with Open-World Generalization

$\pi_{0.5}$

Physical Intelligence
(2025.04, arXiv 2504.16054)

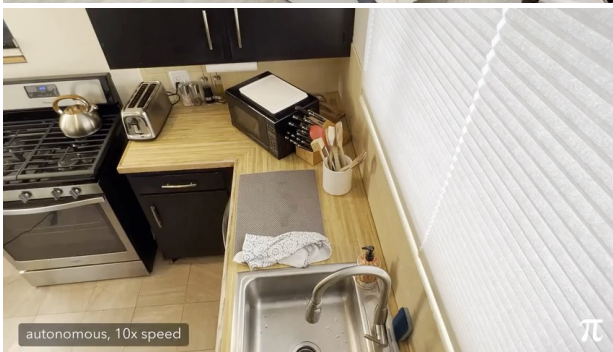
Yejin Yoon

~~2026.09.29~~ 2026.10.20

Motivation: A Home It Has Never Seen

A 10 - 15 minute cleanup task in a never-seen home

$\pi_{0.5}$ walks into a home absent from its training data and completes a 10 - 15 minute cleanup



Instruction

Just one high-level command:
"clean the kitchen"

Environment

No test home appears in the training data

Evaluation

Prior VLAs (incl. π_0) were tested in *training-like* settings

How $\pi_{0.5}$ gets there, step by step from π_0 (+ FAST)

Contents

1

Preliminary

- Robot policy basics $\rightarrow \pi_0$ (flow) $\rightarrow$ FAST (tokens) $\rightarrow$ comparison (recap)

2

The $\pi_{0.5}$ Method

- Training Recipe + Data + Hierarchical Inference

3

Wrap-up

- Summary and two open questions

$\pi_{0.5}$: a VLA with Open-World Generalization

1. Preliminary

1.1 The Problem a Robot Policy Solves

1.2 Naive Tokens: Binning

1.3 π_0 : Flow Matching

1.4 FAST: Compressed Tokens

VLA Inputs, Outputs and Training

VLA = observation $\rightarrow$ next joint positions, at 50 Hz

From images, an instruction and body state, a VLA outputs the next joint positions 50 times a second
It is imitation learning from human demos, structured like LM next-token prediction

Input		Example
image	I_t	One or more cameras
instruction	ℓ	"Pick up the cup"
proprioceptive state	$q_t \in \mathbb{R}^7$	6 joints + gripper = 7 numbers

Output		Example
action	$a_t \in \mathbb{R}^7$	Next joint targets, 7 numbers

Control loop 🧐

Repeats at 50 Hz (1 timestep = 0.02 s).
The model only decides "where to";
a PD controller drives the motors to follow

One example step (joints in rad, gripper 0 $\rightarrow$ 1 = close)

$$q_t = [0.12, -0.40, \dots, 0] \rightarrow a_t = [0.13, -0.38, \dots, 1]$$

	LM	VLA
Context	Previous tokens	Obs. o + instruction ℓ
Predicts	Next token	Next action a
Target	A vocab token	7 real numbers

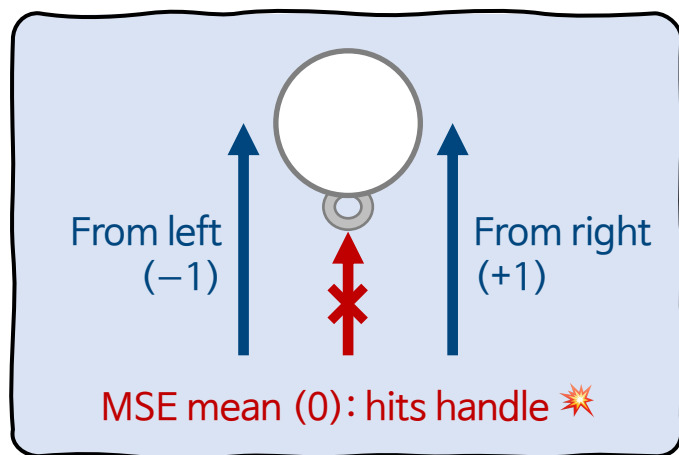
- A human teleoperates the robot for hundreds of demos
- Every 1/50 s, log (image, state, human's target)
 $\rightarrow$ tens of thousands of (o, a) pairs
- Objective: $\max \log \pi(a|o, \ell)$ (behavior cloning)

The target is 7 real numbers, so the obvious loss is MSE, and that causes problem 1

Problem 1: MSE Regression Averages Multimodal Actions

With several valid actions, MSE predicts their mean, which may itself be invalid

When the data contain several valid actions, MSE regression learns their mean, which need not be any of them
LMs avoid this by sampling from a softmax distribution; a robot policy should output a distribution too



Top view · hand approaches from below

Why the MSE solution fails

- Data: half the demos grasp the cup from the left (lateral -1), half from the right ($+1$)
- MSE minimizer = mean = $0 \rightarrow$ moves straight ahead and hits the handle
- Both strategies are valid; their mean is not

NLP

"I ate ____ for breakfast"
 $\rightarrow$ sample 🍏 or 🍌 (50/50),
 not the mean of their embeddings

Robotics

Lateral motion has two modes (-1 , $+1$)
 $\rightarrow$ every sample is one of the valid solutions

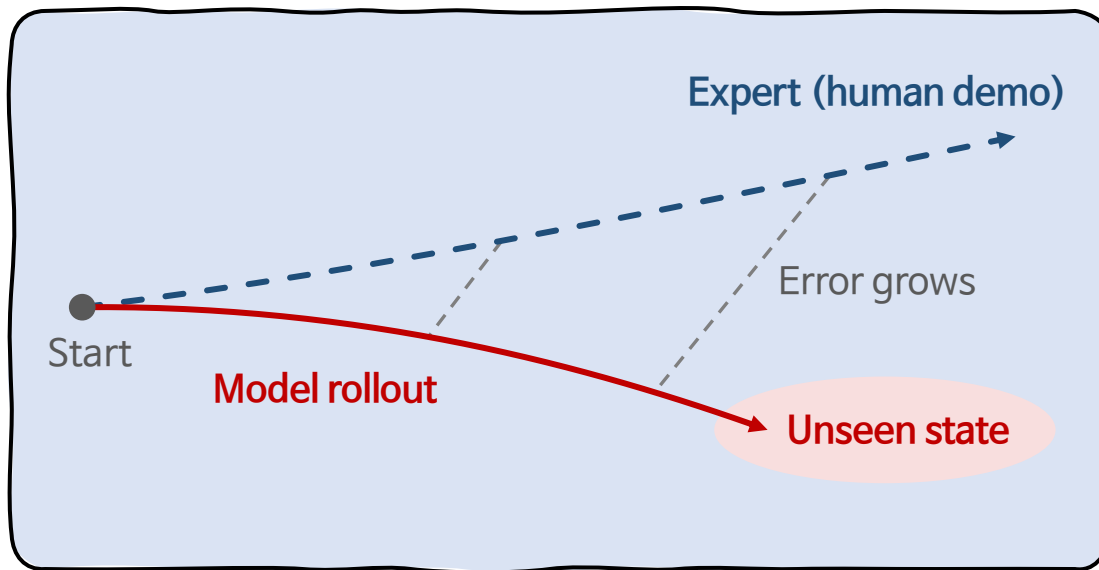
⚠ **Terminology** Multimodal here means **multiple modes**, not multiple input modalities; VLA papers use both senses

A policy should model the action **distribution** and sample from it.

Problem 2: Per-step Prediction Is Slow and Compounds Error

Predicting every 20ms step is slow, and small errors drift into unseen states

Each small error moves the robot toward states absent from the demos, where predictions degrade further;
predicting a whole chunk at once cuts both the calls and the decisions



Trajectories: human demo vs model rollout

Why per-step prediction fails

- **Speed:** a multi-billion-parameter model is too slow to call every 20ms step
- **Compounding error:** small error $\rightarrow$ unseen state $\rightarrow$ worse prediction $\rightarrow$ more drift

NLP

Exposure bias

Train: true prefix

Generate: own prefix

Robotics

Wrong states, not tokens

Train: demo states

Rollout: drifted states

A policy should predict a whole **chunk** of actions at once.

Solution?: Action chunk

Chunking cuts model calls and drift, but MSE still averages the modes

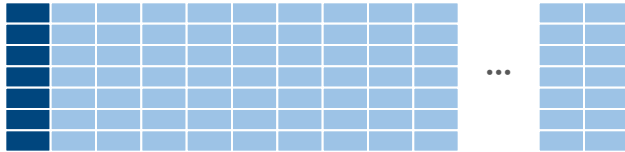
One call predicts 50 timesteps: 1/50 the calls and fewer decisions, but still a mean path

1 step = 7 numbers



× 50

1-second chunk = 50 × 7 matrix



Solves

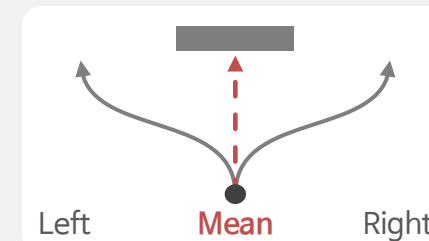
Speed

1/50 model calls

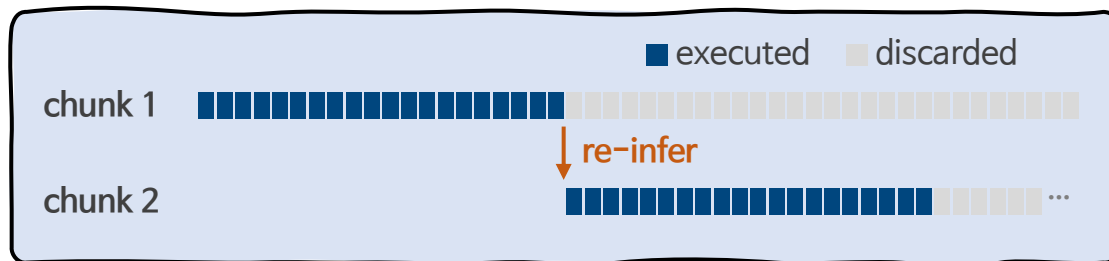
Compounding error

Fewer decisions to get wrong

Does not solve: averaging



50 × 7 regressed with MSE
averages the left and right paths
→ one head-on 1-second path



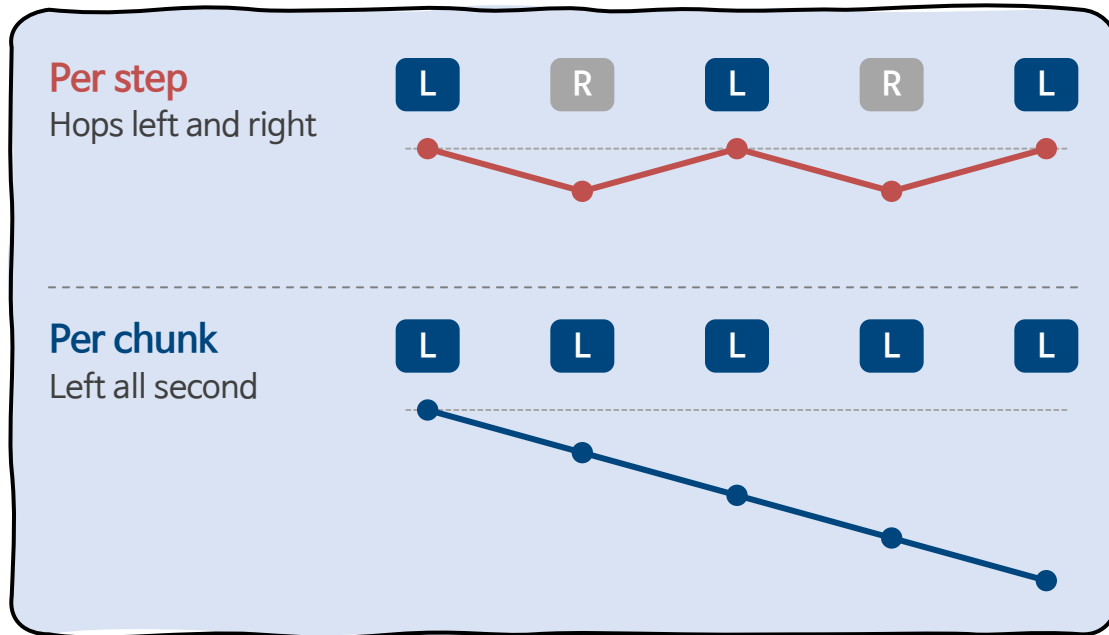
- **Cost:** blind to new observations mid-chunk 🤔

Chunking fixes speed and drift, but averaging still needs a **distribution**

Solution?: Sample from a distribution

Sampling fixes averaging, but per-step samples switch modes

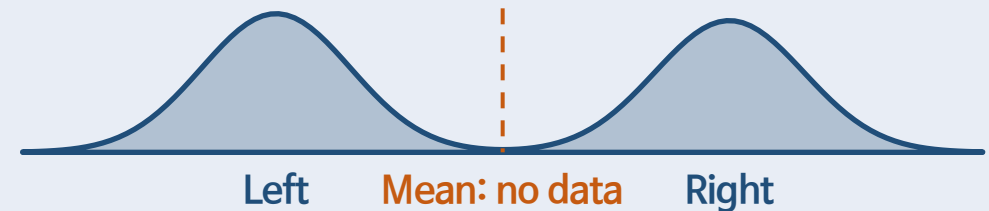
Each step's sample picks a valid mode, but consecutive picks can disagree, so the arm jitters



Arm position (down means left)

- Each pick starts some valid path, but stitched together they miss the cup: **jittery motion**
- **Fix:** sample the 1-second path as **one sample** (temporal consistency)

Two valid modes in one distribution



Distribution and chunking are one design: **a distribution over chunks**

Action Representation in VLAs

Every VLA has to answer this, and there have been **two** ways to do it.

How do we represent and sample a multimodal distribution over a $50 \times d$ real-valued matrix?

(A) Tokens

Turn numbers into tokens, sample with softmax

vs

(B) Generative model

Model the continuous distribution directly

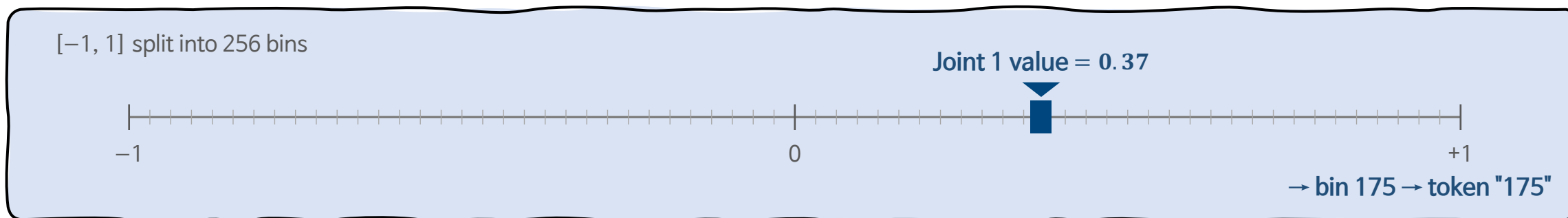


Tokens stall first $\rightarrow \pi_0$ **detours** with flow $\rightarrow$ FAST revives **tokens** $\rightarrow \pi_{0.5}$ uses both

Binning: The Simplest Tokenization

RT-2, OpenVLA: split each value into 256 bins and treat the bin index as a token

Simple and VLM-friendly, but one 1-second chunk becomes hundreds of tokens



One 1-second action = how many tokens?

$$\begin{array}{ccccc} 7 & \times & 50 & = & 350 \\ \text{dims} & & \text{timestep} & & \text{tokens} \end{array}$$

2 arms (14 dims): **700 tokens**

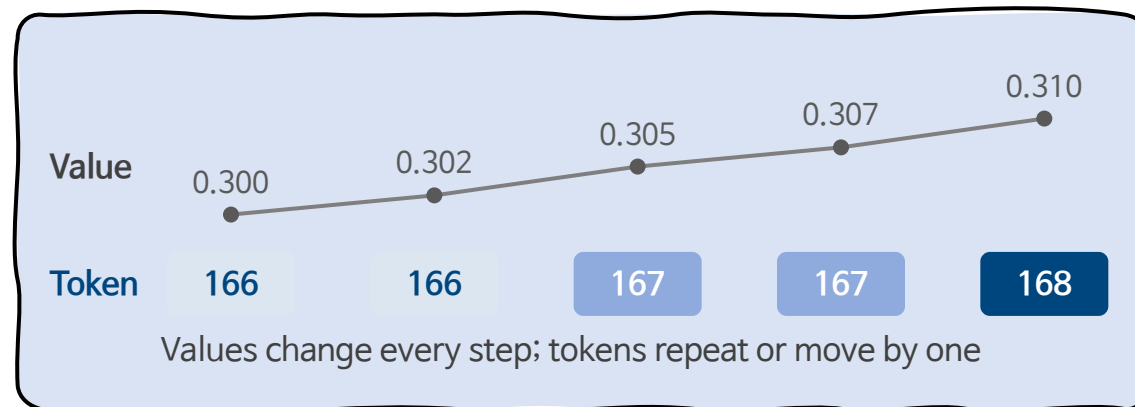
- Pro: a VLM already speaks tokens, so no architecture change
- Softmax models a distribution, and earlier tokens hold the mode
- (+) cross-entropy loss **mixes with text and VQA**

Workable on 5Hz robots, this **breaks down** at 50Hz

Neighboring tokens are near-identical, so the model learns to copy

Slice a smooth path finely and each token nearly repeats the last, so copying wins

- **Teacher forcing**: copying the previous token is mostly right, so easy tokens dominate the loss
- Decisions like "when to turn" or "left or right" sit in a few tokens, so their learning signal is weak
- Higher control rates make neighboring values even more alike



아 진짜 웃겨ㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋ

Blue: must be predicted Gray: copy of the previous character

Copy the last character: right for 13 of the 14 $\Rightarrow$

Easy copies dominate the loss; words are rare

Robot: “when to turn” is where $\Rightarrow$ stops

Tokens stall on precise motion. π_0 drops tokens altogether

π_0 : Fast, Precise, but Costly to Train

Dropping tokens removes binning's failure but adds new costs

$\pi_0(2024.10)$ skips tokenization and generates continuous actions with flow matching

What flow gives



multimodal

Picks one of -1 and $+1$,
no softmax needed



mode switching

The whole 1-second path stays
in one mode



fast inference

All $50 \times d$ values generated
in parallel

What it costs



Expensive training

Far more compute than FAST
for the same performance



Gemma learns indirectly

Only through K, V;
the expert predicts the actions



Awkward to mix

MSE loss does not combine with
web text and VQA (CE)

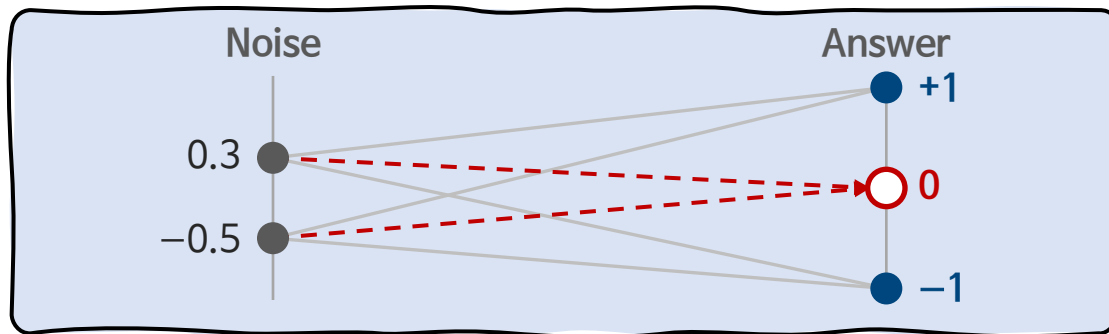
Flow is fast and precise, but **expensive to train** and hard to mix

Flow Matching: Learn a Velocity, Not the Answer

One jump to the answer averages, so flow matching learns a velocity instead

Noise alone can't pick an answer, so connect noise to answer and regress the velocity "answer – noise"

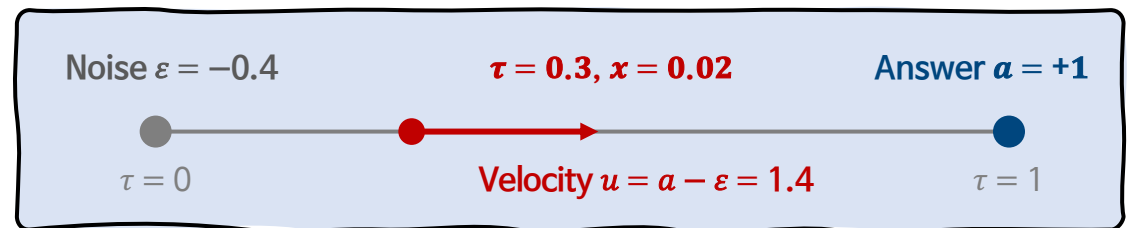
Problem: one jump to the answer



Each noise has two answers, so the output collapses to 0

- A network maps the same input to the same output, so sampling needs random input noise (initial noise)
- The noise is unrelated to the answer, so the model ignores it and learns **the mean 0**

Fix: learn the velocity along a line



- ① Answer $a = +1$, ② noise $\varepsilon = -0.4$, ③ connect them, ④ $\tau = 0.3$ is sampled
- ⑤ Position $x = \tau \cdot a + (1 - \tau) \cdot \varepsilon = 0.3 \times 1 + 0.7 \times (-0.4) = 0.02$
- ⑥ Target velocity $u = a - \varepsilon = 1.4$ per unit time
- **Same ε , other sample:** $a = -1$ gives $u = -0.6$

Regression
one mean for all $\rightarrow 0$

Flow matching
mean given position $\rightarrow$ one answer

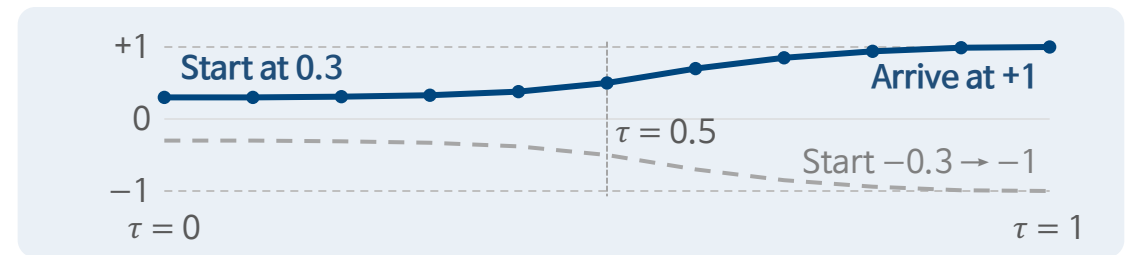
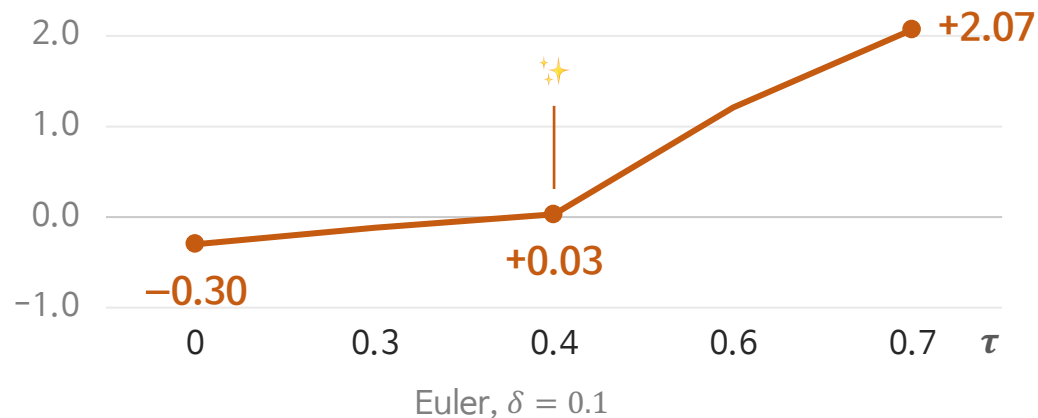
Still MSE, but seeing the current position lets the **velocity** avoid the mean

Inference: Follow the Velocity from $\tau = 0$ to $\tau = 1$

Unsure at first, then it commits: the endpoint is always one answer

Start from noise at $\tau = 0$ and move in small increments ($\delta = 0.1$) until $\tau = 1$; the starting noise decides the side

Velocity along the path from 0.3, by τ



Why small moves? Arrival position by move size δ

0.00	0.29	1.00	1.00
$\delta = 1$	$\delta = 0.5$	$\delta = 0.2$	$\delta = 0.1$

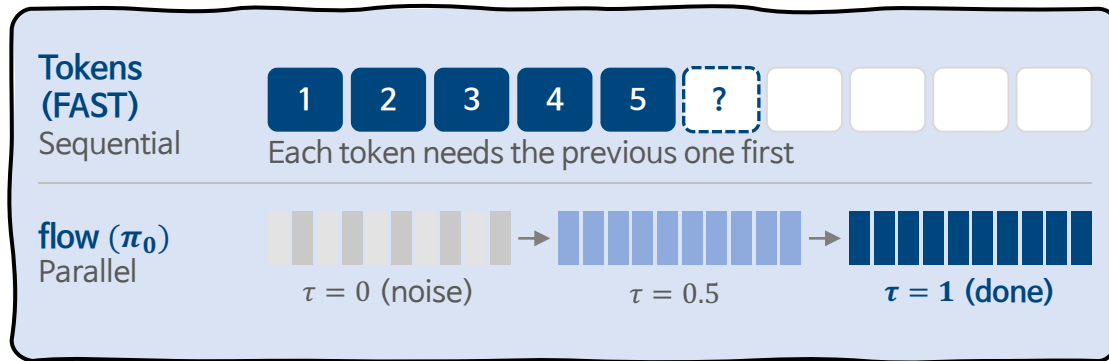
- From -0.3 it lands at -1.00 : the mean exists only in transit, the endpoint is always a peak
- $\delta = 0.1$ is an empirical accuracy/speed trade-off (no ablation in π_0 or $\pi_{0.5}$ 🤔)

Small moves along τ reach a **peak**; one jump from $\tau = 0$ to 1 averages

Scaling Up: The Whole Chunk Moves at Once

Same idea at full size: one noise draw sets the whole chunk's mode

350 numbers (50×7) move along their own lines at once, like the 1-D example repeated



Robot time (timestep)

Range

0 - 0.98s, 50 slots

Relation to tokens

Spread across 50 tokens

Flow time (τ)

Range

0~1, $\delta = 0.1$ 10 denoising steps

Relation to tokens

Shared by all tokens

- Every slot is refined in every pass; none waits
- Tokens wait for the previous one;
flow has every slot from the start (10 parallel passes)

NLP counterpart: diffusion LMs

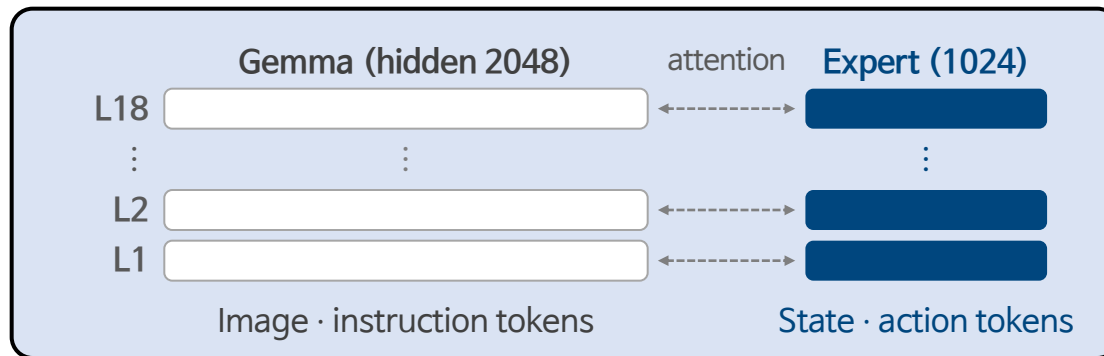
A diffusion LM (e.g., LLaDA) unmask **all words** over a few passes; flow does this with continuous values

The whole chunk is refined **in parallel**, so no mode switching

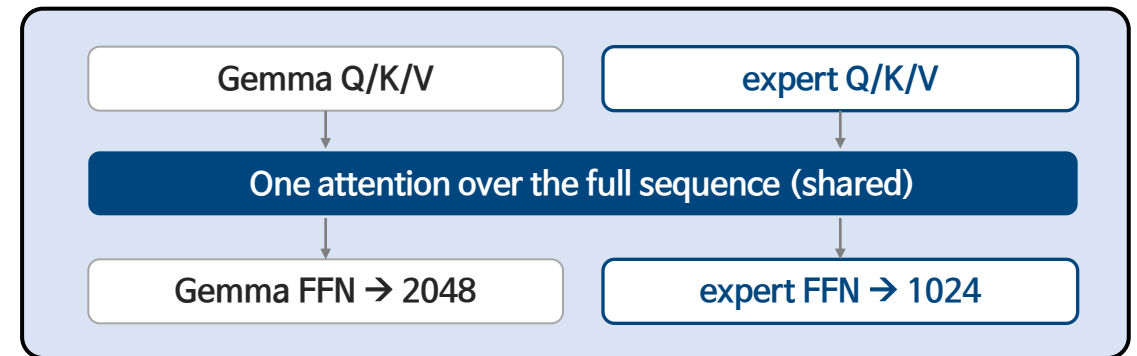
π_0 Architecture: VLM + Action Expert

PaliGemma (3B) + action expert (300M), paired in each of 18 layers

π_0 is one 18-layer transformer: PaliGemma (3B) with an action-only expert (300M) paired into every layer



- Read the scene and instruction (VLMs already can), and compute velocity from noisy actions (**Expert**)
- Not two transformers merged at the end, but **merged in attention at every layer**
- After 18 layers: 50 action tokens $\rightarrow$ linear $\rightarrow$ velocity



Tokens in one sequence

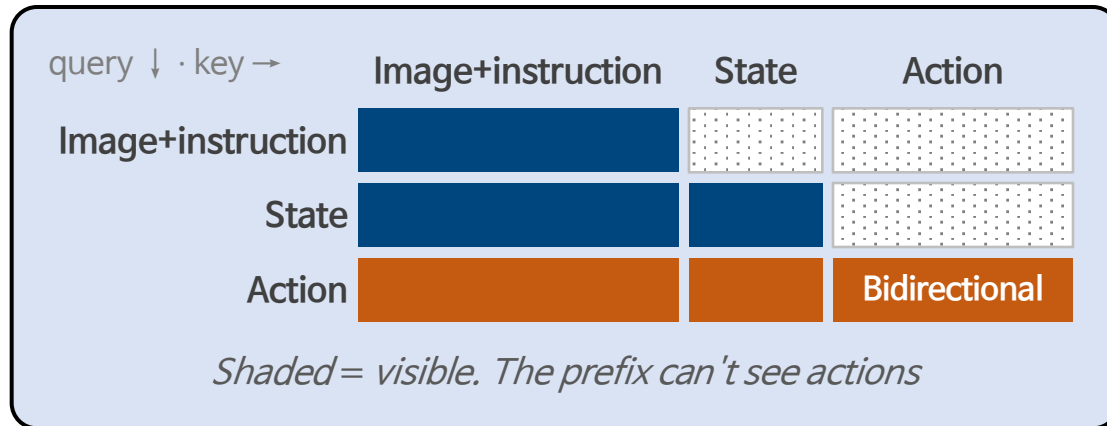
Image	Instruction	State	Action
256 per 📷	Variable	1	50
Gemma weights · hidden 2048		Expert weights · 1024	

The VLM and the action expert meet through **attention at every layer**

Attention Mask and KV Cache

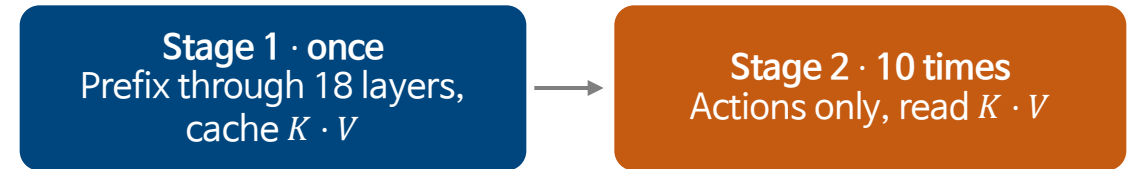
Block the static side: one heavy pass + ten light passes

Hide the actions from the static prefix, compute it once and cache it: π_0 runs one heavy pass + ten light ones



- [image | instruction | state] is fixed across all 10 steps; **only [50 actions] change**
- If the prefix saw actions, it would be recomputed every step, so **block it**

KV cache



Same idea as LLM prefill

Heavy Gemma passes

1

π_0 (flow)
+ 10 light expert passes (parallel)

30 – 50

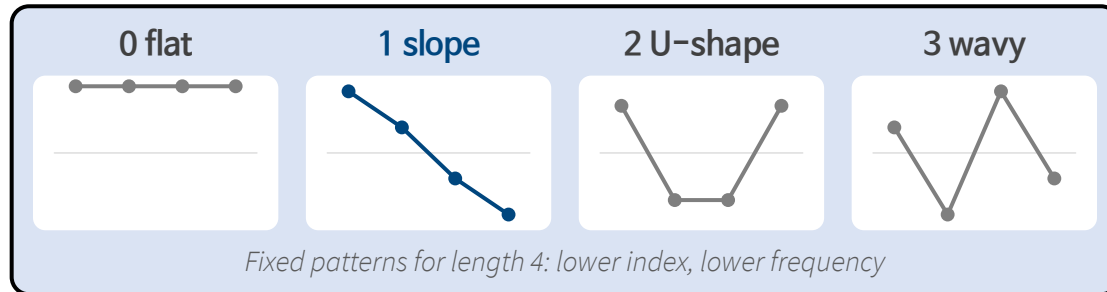
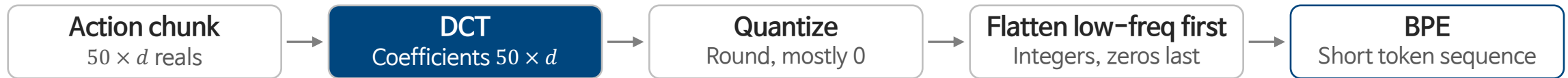
Token approach
Sequential, one token each

The prefix is **cached**, so each pass recomputes only the actions

FAST: DCT Packs a Smooth Path into Low Frequencies

DCT rewrites a path as a weighted sum of fixed wave patterns, like JPEG

FAST swaps look-alike neighbors for a decorrelated basis, DCT (recap of Alexis's session 1 talk)

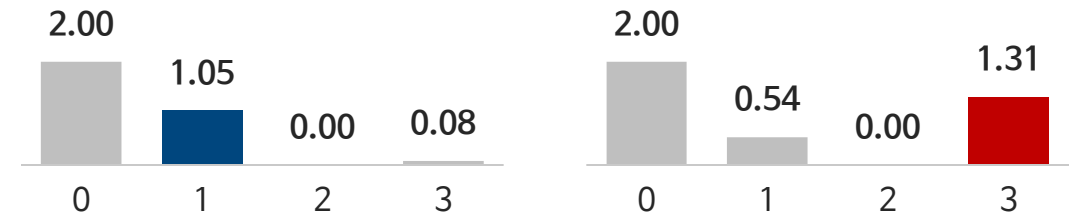


- Coefficient = dot product of the path and a pattern
- Like query · key scores against fixed keys
- In practice: length 50 per joint, 50 coefficients

DCT coefficient magnitude (absolute), length-4 path

Smooth [0, 0.33, 0.67, 1]

Jagged [0, 1, 0, 1]



- Smooth paths land in 1 (low freq), jagged ones in 3 (high freq); robots are the first case
- Orthogonal patterns: each coefficient covers a different aspect, so none can copy another

Robot paths are smooth, so DCT keeps only a few **low-frequency** coefficients

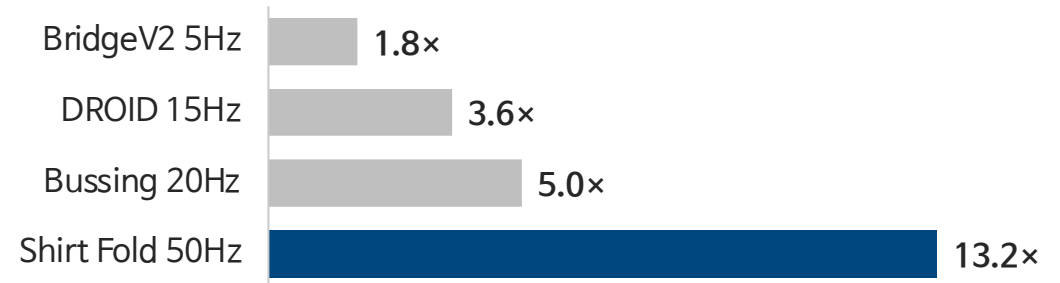
FAST: 700 Tokens Become 53, but Generation Is Sequential

Training compute drops, but inference must draw tokens one by one

Rounding, flattening, and BPE turn 700 tokens at 50Hz into 53

- **Quantize** (scale $\times$ round): values $\times 5 \rightarrow 10, -5, 0, 0$
The scale trades precision for compression;
- **Flatten**: every joint's index $0 \rightarrow 1 \rightarrow \dots$
[10, 10, -5, 5, 0, 0, 0, 0]: coarse motion first, zeros last
- **BPE**: frequent "0 0 0 0" becomes one token
 $\rightarrow$ 8 tokens become 5

Compression ratio (naive $\div$ FAST tokens)



Shirt Fold: 700 $\rightarrow$ 53 tokens; higher rates compress more

$\sim 5\times$

Result · training compute saved

Performance close to π_0 , better language following

30 – 50

Cost · one token at a time

30 – 50 tokens per chunk, each a full VLM pass; next-chunk prep is tight

So the two branches end up with opposite weaknesses

$\pi_{0.5}$: a VLA with Open-World Generalization

2. $\pi_{0.5}$

2.1 Training Recipe

2.2 Data

2.3 Hierarchical Inference

$\pi_{0.5}$ at a Glance

Pretrain with tokens, add flow in post-training, infer with flow only

The action expert does not exist until post-training, where it is added with random initialization

Pretraining		→	Post-training		→	Inference	
Model	Gemma only (no expert)		Model	Gemma + expert (random init)		Model	Gemma expert
Action	FAST tokens		Action	FAST + flow		Action	flow only (no FAST)
Steps	280k		Steps	80k		Steps	—
Loss	CE		Loss	CE + 10 × MSE		Loss	—
Data	Robots, web, subtasks		Data	Filtered robot + web		Data	—
Output	Text + FAST tokens		Output	Text + flow actions		Output	Subtask, then actions

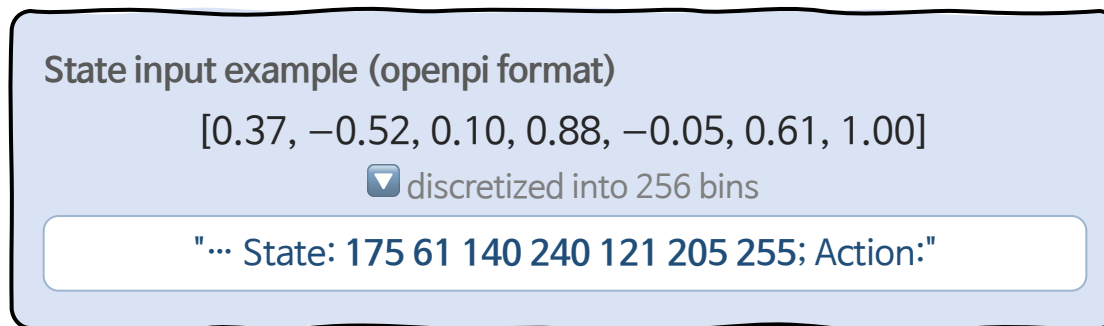
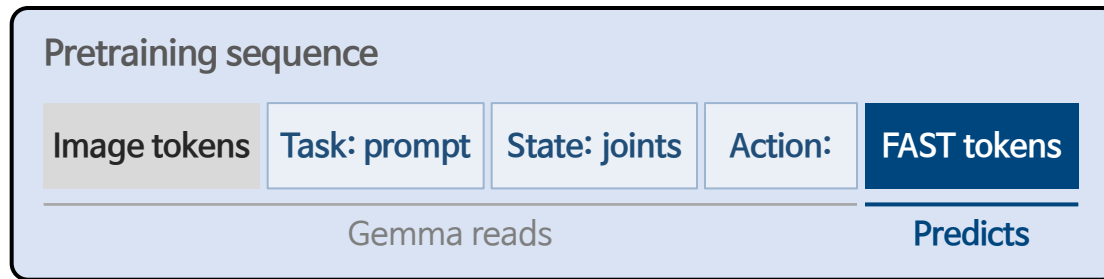
- Cheap, easy-to-mix training comes from tokens; fast, precise inference from flow

Train with **tokens**, infer with **flow**: cheap to learn, fast and precise to run

Pretraining: Gemma Learns Actions Directly

Without an expert, Gemma predicts FAST tokens itself

It also reads the robot state as plain text tokens



π_0

Action
flow

Predicted by
expert (Gemma indirect)

State input
Continuous (expert)

$\pi_{0.5}$ pretraining

Action
FAST tokens

Predicted by
Gemma directly

State input
Text tokens (Gemma)

- Binning is fine for inputs: slide 11's problem came from 50 near-identical output timesteps; state is one moment

Gemma now learns actions directly, and with everything as tokens, state comes along as text

Post-training: Attach the Action Expert

The same answer is learned in two forms, FAST and flow, at once

Post-training adds a new expert and narrows the data to the target domain

Purpose 1: Attach the flow expert

flow is trained only in the last 80k steps

Purpose 2: Narrow to the target task

97.6% of pretraining data is not the target robot;
same as LLM pretraining $\rightarrow$ SFT

π_0 : input, once

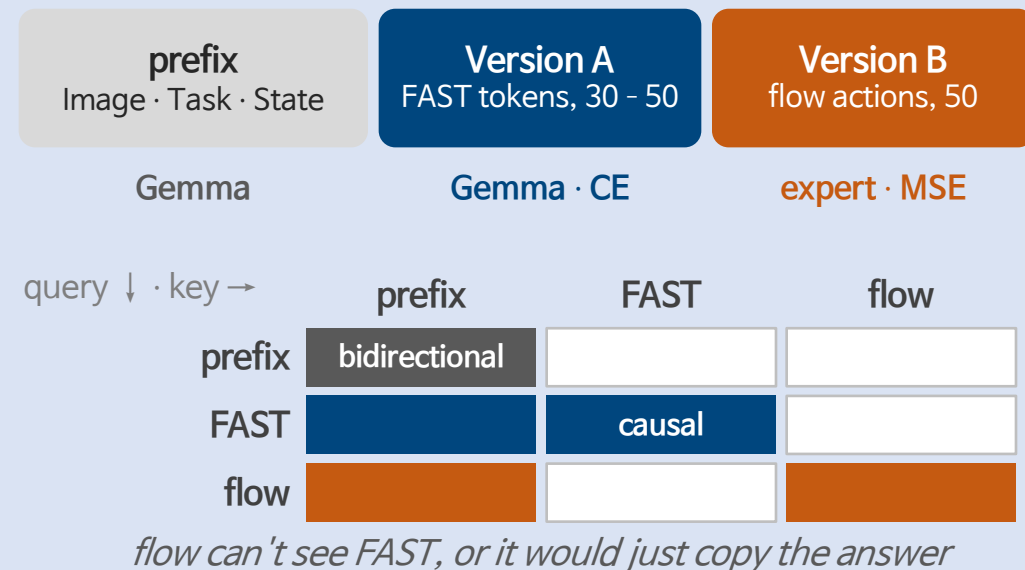
$\text{MLP}([W \cdot a; \varphi(\tau)])$

$\pi_{0.5}$: every layer

$\varphi(\tau) \rightarrow \text{MLP} \rightarrow \gamma$ of RMSNorm

- loss = CE (text + FAST) + $\alpha \times$ flow MSE
Pretraining $\alpha = 0$, post-training $\alpha = 10$
- Why CE stays on (paper): it preserves text prediction, vision, and semantics. No ablation with CE off

The same answer twice in one sequence



The same answer appears twice in one sequence, so the two versions must not see each other

Inference: No FAST

Subgoal text → flow actions; FAST is only a training auxiliary

At inference the model writes a subgoal first, then generates actions with flow

- 1 **Subgoal** Gemma (autoregressive)
[4 cameras | Task: clean the kitchen, State: ...]
→ "pick up the plate"
- 2 **Action** Gemma prefix once + expert refinement
[2 wrist + front camera | Task: pick up the plate, ...]
→ 1-second action (50×18)
- 3 **Execute** 50Hz, PD control
If the situation changes, ① becomes the new subgoal

- FAST's benefit stays in Gemma's weights, not in tokens: Gemma builds the prefix K, V that the expert reads
 - "FAST is **slow** at inference"??:
the slow part is used only in training
- The paper does not say when the chunk is regenerated (after all, after part, or during execution)

	Training	Inference
FAST tokens	Pretrain + post-train	Not used
Flow expert	Post-training only	Used

FAST helps only in training; at inference it is a subgoal followed by flow

Experiment: Comparison with π_0 (Q4)

$\pi_{0.5}$ wins clearly: read it as a training-method effect plus a data effect

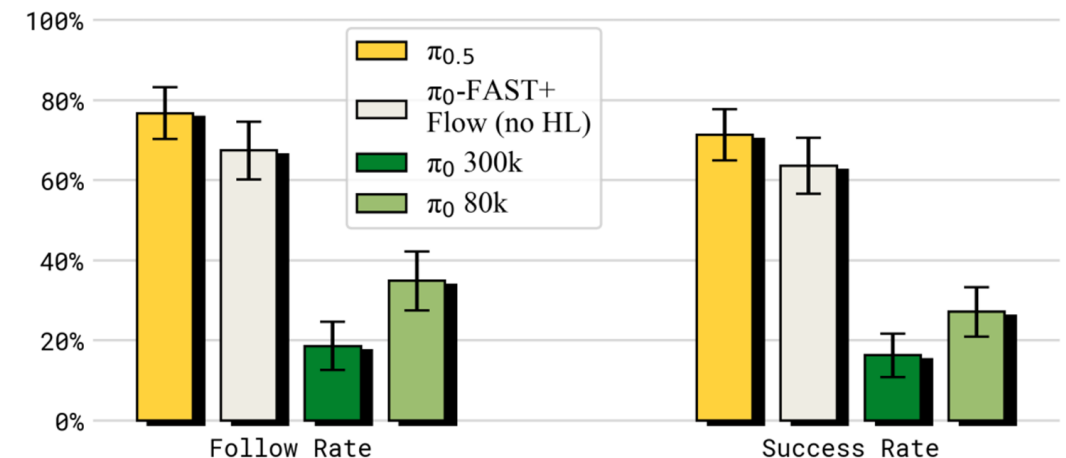
On the same robot data, $\pi_{0.5}$ beats π_0 by a wide margin; the gap splits into a method effect and a data effect

average task progress



- $\pi_{0.5}$ wins, even against π_0 at 300k steps

language following



$\pi_{0.5} > \pi_0$ -FAST+Flow (slightly) $\gg \pi_0$

Likely learned mainly from token training

The recipe makes mixing possible; what gets mixed drives generalization to new homes

Six Data Sources

97.6% of pretraining data is not the target robot

Only 2.4% comes from the target robot; each of the six sources teaches something different

Code	What	Pre	Post
MM	Mobile Manipulator: target robot, ~100 homes, ~400 hours	✓	✓ short successful episodes only
ME	Multi-Environment: fixed arms taken to many homes (environment diversity)	✓	✓ filtered
CE	Cross-Embodiment: lab robots + OXE (simple scenes, diverse tasks)	✓	✗
HL	High-Level subtask: subgoal text + object boxes per segment	✓	ME only
WD	Web Data: captions, VQA, object detection + indoor boxes	✓	✓
VI	Verbal Instruction: humans directing the robot in words	✗	✓ new

- Groups: robots with actions (MM, ME, CE) / labels on robot data (HL, VI) / data without actions (WD)

VI: humans pick doable commands in states the policy actually reaches (DAGger, cf. slide 6)

Policy runs on the robot



Expert speaks a command



Used as subgoal example (~11%)

Even data without actions is usable because $\pi_{0.5}$ splits action into two stages

Splitting the Action Divides the Data Work

Separate "what" from "how" and each data type finds its use

Split "*what to do*" from "*how to do it*", and data that robot learning could not use becomes useful on the matching side



- **low-level** only needs "*pick up the pillow*", so it learns from **other robot bodies** too
- **high-level** only decides what to do, so it learns from **data without actions**
- Note: one model learns both, so the split is not complete (web vision knowledge can help low-level)

high-level (what)

Skill needed

Meaning, vision

Data taught

HL, VI, WD

Needs action data?

No

low-level (how)

Skill needed

Physics

Data taught

MM, ME, CE actions

Needs action data?

Yes

Merging different bodies

zero-padding the rest is 0

normalization per dataset, -1 to 1

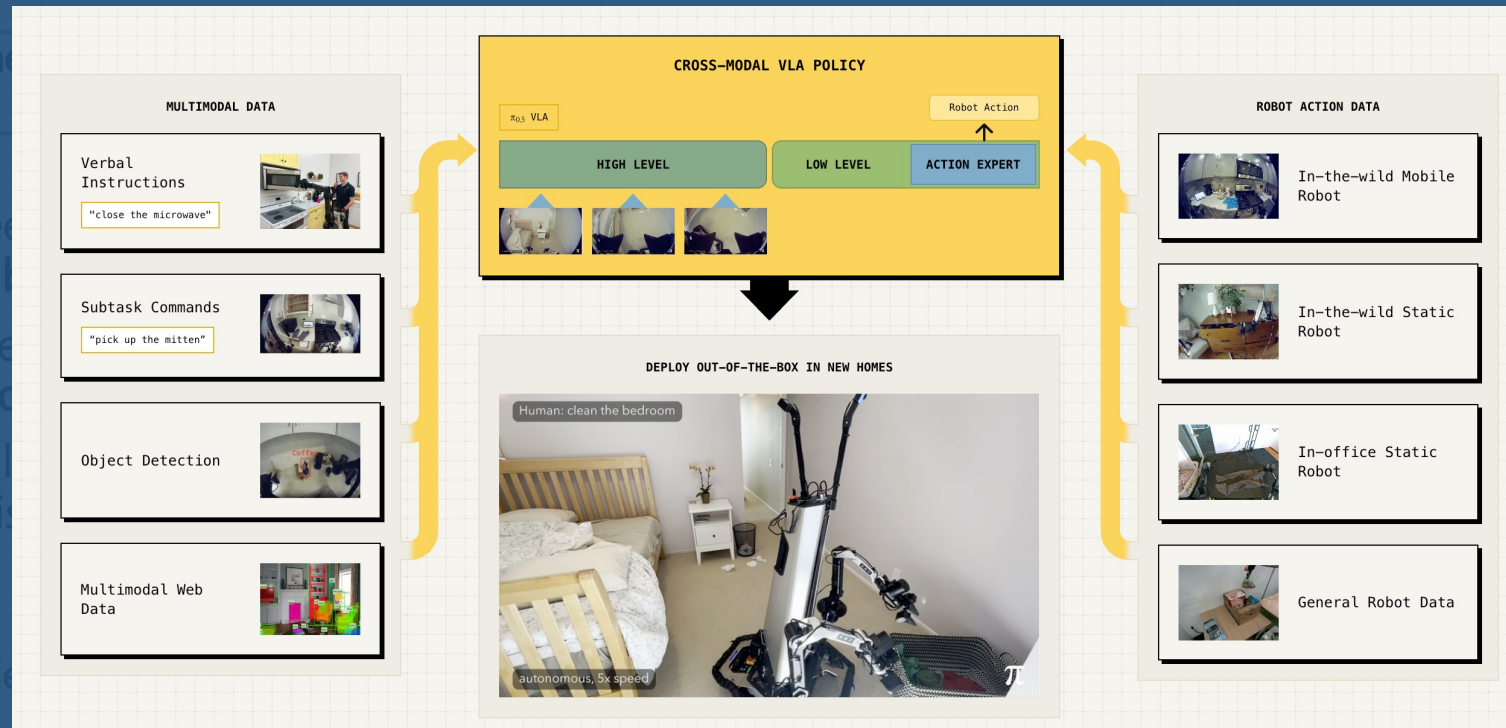
control-mode token joint / end effector

Splitting the action divides the data work, and robots with different bodies merge easily

Splitting the Action Divides the Data Work

Separate "what" from "how" and each data type finds its use

Split "*what to do*" from "*how to do it*", and data that robot learning could not use becomes useful on the matching side



Splitting the action divides the data work, and robots with different bodies merge easily

The Same Structure in One Model

Hi Robot's decomposition, in one model with one loss

$\pi_{0.5}$ uses Hi Robot's "subgoal $\rightarrow$ action" decomposition, but one model learns both steps with the same loss

Decomposition

$$\pi(a, \hat{\ell} | o, \ell) = \pi(a | o, \hat{\ell}) \times \pi(\hat{\ell} | o, \ell) \quad (\text{chain rule})$$

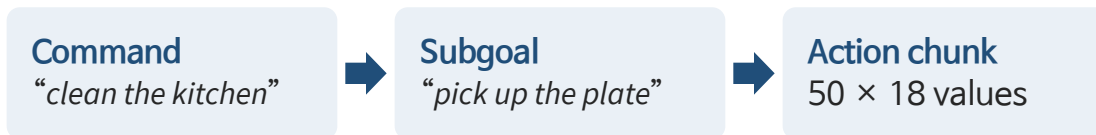
- o observation / ℓ high-level command "clean the kitchen" / $\hat{\ell}$ subgoal the model generates "pick up the plate" / a action chunk 50×18
- A full chain rule would be $\pi(a | o, \hat{\ell}, \ell)$; ℓ is left out on purpose $\rightarrow$ actions see only the **subgoal**
 $\rightarrow$ low-level becomes a reusable skill executor

📌 NLP counterpart: CoT

It generates an intermediate step (one subgoal line) first, then answers (actions) conditioned on it

	Hi Robot	$\pi_{0.5}$
Model	VLM + policy	One
Training	Separate	Joint loss
Sharing	Linked by text only	Gemma weights

- Training data: high-level uses HL and VI;
low-level uses robot actions labeled with subgoals



1 model, same weights, same loss 😊

Next: which subgoals did it produce in a real, unseen home?

A Real Inference Example

One high-level command → subgoals in order

In an unseen home, one command yields subgoals in order: wide view for "what", hands only for "how"

High-level command	Generated subgoals
<i>put the items in the drawer</i>	pull out the drawer → pull out the top right drawer → pick up tong → put tong into drawer → push the top drawer
<i>place the dishes in the sink</i>	pick up plate → put plate in the sink → put cup in the sink → pick up the spoon → pick up bowl
<i>put the laundry in the laundry basket</i>	pick up shirt → put the shirt in the laundry basket → put clothes in laundry basket

	Cameras	Output	Frequency
high-level	All 4 (front, back, both wrists)	A few tokens	Lower frequency
low-level	2 wrists + front	50 × 18	Every chunk

- Side effect:
low-level is trained on subgoal-level commands,
so a person can give detailed ones like "*pick up the plate*" directly

Home 1

Human: "put the items in the drawer"

HL prediction:



Home 2

Human: "place the dishes in the sink"

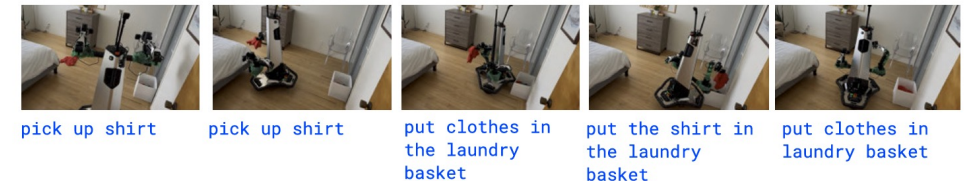
HL prediction:



Home 3

Human: "put the laundry in the laundry basket"

HL prediction:

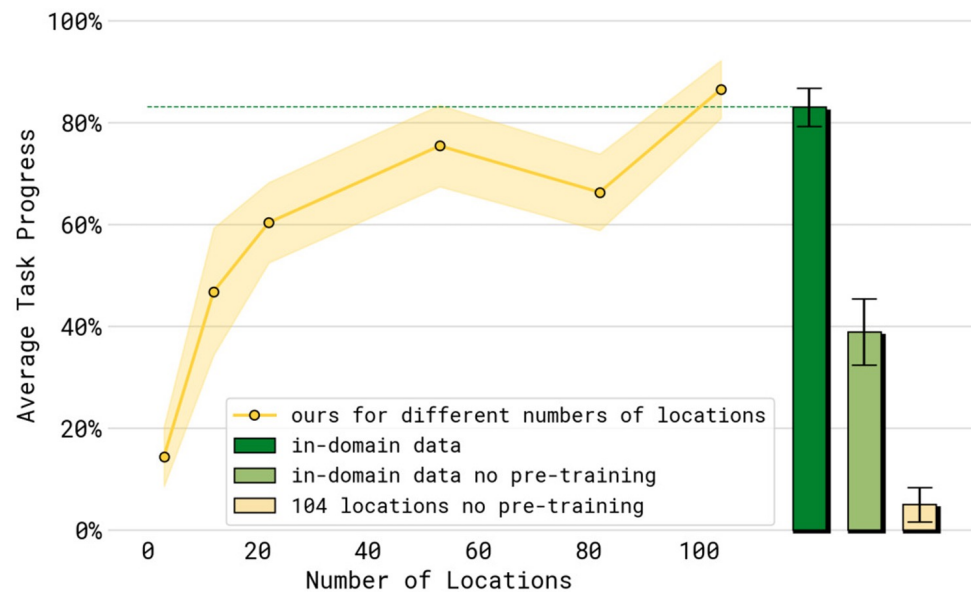


One command is enough: the **model writes its own subgoals** in a new home

Experiment: Number of Training Homes (Q2)

104 training homes match a model that saw the test home

A model that never saw the test home matches one that did, once it trains on 104 homes; seeing the test home without other data stays far lower



Model	Test home	co-training	Performance
104-home model	unseen	yes	high
Dark green	seen	yes	$\approx$ 104 homes
Light green	seen	no	much lower
Light yellow	unseen	no	much lower

- **Setup:** MM homes 3, 12, 22, 53, 82, 104; steps fixed at 40k to equalize unique samples
→ the curve shows environment diversity

* For cost, only post-training varies on a pretraining w/o MM; (not identical to the actual recipe)

Diverse environments plus other data matter more than having seen that home

Experiment: Leaving One Data Source Out (Q3)

Web helps on new categories, other robots help everywhere, ME $\succ$ CE

Web helps on unseen categories, other robots help everywhere, and environment diversity beats task diversity

	ID follow	ID success	OOD follow	OOD success
$\pi_{0.5}$	86	83	94	94
no WD	86	82	80	74
no CE	74	67	67	49
no ME	66	57	33	31

- **Setup:** 2 unseen kitchens, put the instructed one of 5 objects.
ID = new objects from seen categories
OOD = unseen categories (funnel, pill bottle, safety glasses)
- **Overall (Fig 10):** dropping ME or CE hurts a lot
(* no WD is not significantly different)

WD Web data

ID unchanged (83 $\rightarrow$ 82), OOD drops sharply (94 $\rightarrow$ 74)
 $\rightarrow$ vision and semantics

ME, CE Multi-environment · Cross-embodiment
ID and OOD both drop $\rightarrow$ physics, across the board

ME $\succ$ CE

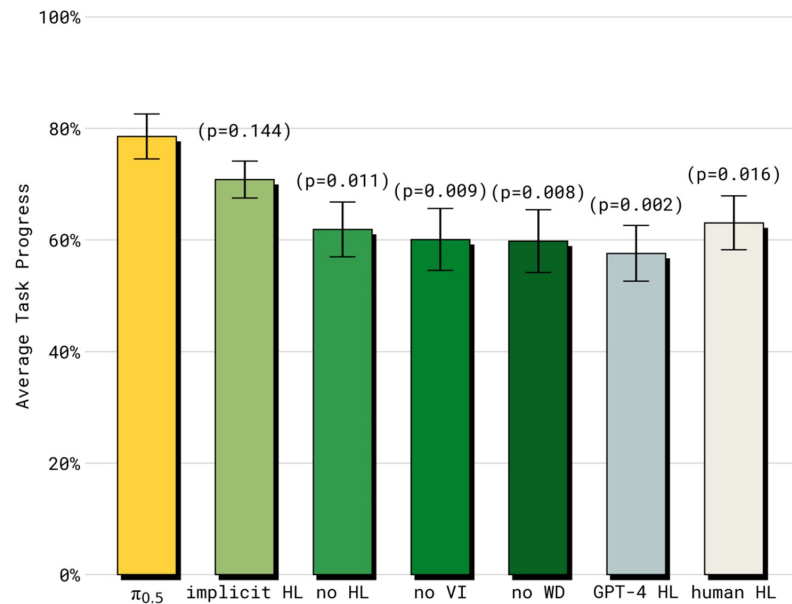
OOD success 31 vs 49
 $\rightarrow$ environment diversity $\succ$ task diversity

Other robots matter everywhere, web data on new categories, environment diversity most

Experiment: Changing the High-Level (Q5)

Explicit subgoals are best, and predicting them only in training already gains a lot

Explicit generation is best, but just predicting subgoals in training keeps most of the gain;
GPT-4 is last, and $\pi_{0.5}$ even beats humans



no HL: no subgoals · no WD: no web data

Implicit HL ranks 2nd training only, not at inference

Predicting subgoals helps more as an auxiliary task that refines representations than as an inference step, like CoT-trained models keeping performance without CoT

VI 11% VI removed from training

Removing it hurts a lot: the "doable command" property

GPT-4 ranks last GPT-4 writes the subgoals

What it lacks is not general intelligence but a sense of which commands this low-level can carry out

Higher than the human oracle expert writes the subgoals

(No analysis in the paper 🤔)

That covers $\pi_{0.5}$'s claims and evidence. Next: how far to trust them, and what is unsolved

$\pi_{0.5}$: a VLA with Open-World Generalization

3. Wrap-up

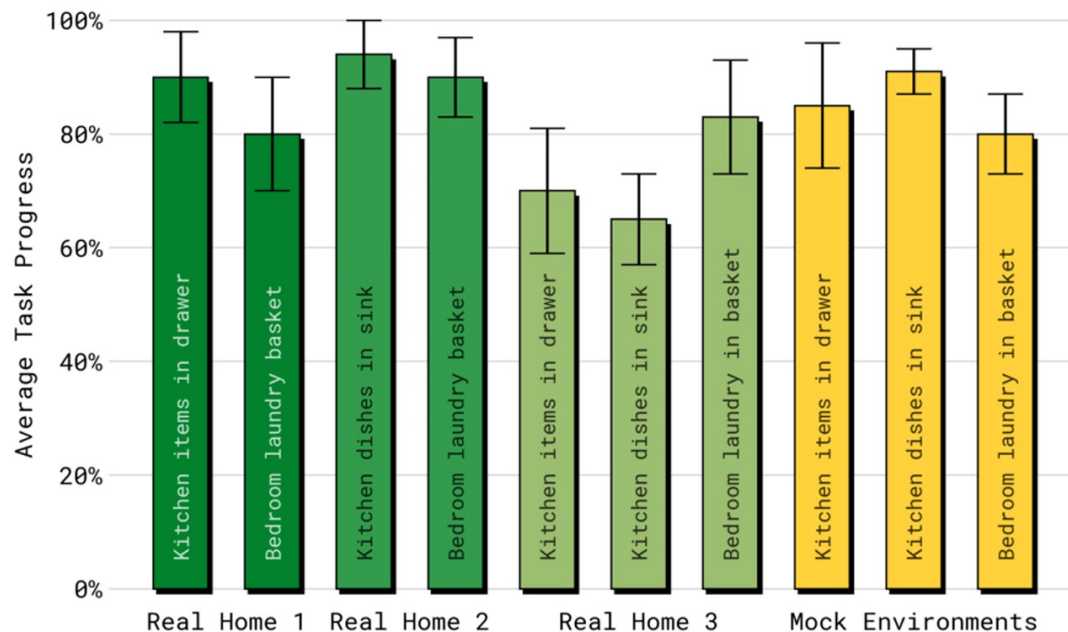
Real-Home Results

Summary

Real-Home Results and Limits of the Experiments (Q1)

Consistent success in 3 real homes, but read the numbers with care

Real-home success supports the mock-home comparisons, but few trials and baselines call for caution



- **Unseen places only:** lab-staged mock homes (for comparisons) + 3 real homes (final check)
- **4 tasks**, partial scores: Dishes in Sink 8 / Items in Drawer 4 / Laundry Basket 3 / Make Bed 5
- **Q1:** mock performance represents real performance well

Limitation	Detail
Trials	10 per task, 40 per policy
Metric	Partial score, not full success rate
Baselines	Own models only
Open data	Not released
Scaling	Pretraining differs from this recipe
Sources	<i>Blog numbers for language following only</i>

Real homes confirm it, but few trials and no released data limit the claims 🤔

Summary

Learn with tokens, act with flow, generalize by mixing data

$\pi_{0.5}$ uses tokens and flow at different stages, and the data mixing that allows generalizes to new homes

Train	Long pretraining uses cheap, easy-to-mix tokens (FAST, CE); inference uses fast, precise flow
Data	Actions split into "what" and "how", with data matched per level (other robots, web, subgoals, VI). 97.6% of pretraining is not the target robot
Result	With 104 training homes, a model that never saw the test home does as well as one trained on it. Predicting subgoals in training matters more than generating them at inference.

Q1. Why does $\pi_{0.5}$ beat the human oracle?

People name the logical next step,
the model names steps this policy handles well;
finer subgoal timing; human input and reaction limits.

Q2. Is the subgoal an inference step or a training signal?

Implicit HL ranks 2nd, no HL drops a lot, and the 1st - 2nd gap
is the extra gain of explicit inference.
Same as the CoT debate in LLMs

Diverse co-training lets one model do 10 - 15 minute cleaning tasks in unseen homes

Thank You

Yejin Yoon

HYU NLP Lab.

Dept. of Computer Science
Hanyang University, South Korea

stillwithyou@hanyang.ac.kr