

Robotensor Competition:

An Open Frontier for Adaptive Robot Intelligence

Robotensor

October 2026

Abstract

Robots meet work that keeps changing: new products, objects and layouts. The usual answer is to collect data and retrain. Robotensor Competition pursues the alternative, adaptive robot models that learn new work from what they are shown, at deployment, with their weights frozen, and it runs as a Bittensor subnet that pays the miners whose training produces the most adaptive model.

We define adaptive robot intelligence as performance after conditioning on context, with frozen weights. We lay out the two axes along which a test can demand adaptation, what changes between context and test and what the context is, and we separate recognizing a known task from learning a new one. The first competition, **Robotensor Vector**, measures action-aware in-context adaptation: a policy is shown one robot demonstration of a task and must perform it where the objects have moved and the acting arm may differ.

Every entrant trains the same 781M-parameter in-context policy, **vector_v1.1**, built on the Behavior Prompting Policy architecture, so the competition compares adaptive model training (data, curricula and methods) rather than model size. Validators settle entries in king-of-the-hill duels on units dealt from a chain block finalized after the entrant commits. No miner code runs, every duel is published, and any duel's units can be re-derived from public values.

Around the mechanism we give a threat model and integrity rules for copied weights, unavailable repositories and Sybil entries, and a measurement protocol, including an adaptation gain that tests whether a model reads its context at all.

Executive summary

Robotensor Competition is a Bittensor subnet that pays for adaptive robot models: models that learn new work from what they are shown, at deployment, with their weights frozen. Its first competition, Vector v1, shows a policy one demonstration of a task and scores it on doing that task in a scene it has never seen. Every miner trains the same in-context network, built on the Behavior Prompting Policy architecture, and validators crown the best model in king-of-the-hill duels on scenes drawn from a chain block finalized after the entrant commits. No miner code runs, and every duel is published.

At a glance

Competition	Vector v1: one robot demonstration, a new scene
Tasks	16 tasks on two Franka arms: 14 pick-and-place, 1 press/push, 1 insertion
Model	vector_v1.1 : 781M parameters, weights only
Duel	160 paired units (16 tasks $\times$ 10), seeded after the commitment
Crown rule	Challenger’s average success beats the king’s by at least 3 points
Rewards	30% of miner emissions to Vector, shared 40 / 30 / 20 / 10% by the four newest champions
Submissions	One per hotkey; a Hugging Face repository holding only model.safetensors
Roadmap	Wider scene variation one factor at a time, new skill categories, cross-robot evaluation, Robotensor Horizon

1. Introduction

Automation works until the work changes. A new product, object or layout still sets off the same cycle: collect data, retrain, reprogram, reintegrate. Pretrained robot foundation models shorten that cycle with broad physical priors, but most of them still meet new work by fine-tuning, which means a new set of weights for every change.

Adaptive models change what gets deployed. An in-context robot learner keeps its weights frozen and adapts within a single deployment from the context it is given: a demonstration, a video of a person doing the job, or its own recent experience. Whatever it works out lives in its context and is forgotten when the context is cleared, so its adaptation is a property of the very forward pass the robot runs in the field. Research has been moving this way, from one-shot imitation (Duan et al., 2017) to in-context imitation with transformers (Fu et al., 2025; Di Palo & Johns, 2024) and video-conditioned policies (Jain et al., 2024).

Adaptation is hard to measure. Benchmarks fix their tasks and test sets in advance and randomize the scene far more than the task. A model that recognizes a familiar task can pass for one that learned a new one, and static test sets leak into training over time. A measurement of adaptation needs four things: frozen weights, contexts and test conditions the model could not have seen, a context the model cannot do without, and the same test for every model compared.

Robotensor Competition builds that measurement into an open incentive market:

- **One fixed architecture.** Every entry is a set of weights for the same in-context network, so a win reflects adaptive model training (data, curricula, objectives) rather than a larger model or a different interface.
- **Evaluation after commitment.** Each duel’s scenes are drawn from a chain block finalized only after the challenger commits its weights, so no one can tune for the test.
- **Public, reproducible results.** Every duel’s units, scores and clips are published, and anyone holding a record can deal the same units again.

Bittensor supplies what such a market needs: a ledger for commitments, a source of shared randomness, and an emission schedule that pays the miners validators agree are best.

Contributions

1. **A problem setting for adaptive robot intelligence.** Performance after conditioning on context with frozen weights; two axes of adaptation; the line between recognizing a task and learning one; and an adaptation gain that tests whether a model uses its context at all (section 3).
2. **A competition that rewards adaptive model training.** Miners submit weights for one pinned in-context network, never code, and a king-of-the-hill duel decides which training produced the more adaptive model (sections 5 and 6).
3. **A reproducible, unpredictable evaluation protocol.** Units are dealt from a duel’s id and a block finalized after the commitment; both sides play the same scenes from the same demonstrations; every record can be checked against public values (sections 6 and 8).
4. **Vector, a benchmark for action-aware in-context adaptation,** that diversifies its scenes one factor at a time and grows its task set by skill category (section 4).
5. **A threat model and integrity rules** for copied weights, unavailable repositories and Sybil entries (section 7).

2. Related work

Robotensor Competition sits where four lines of work meet: learning to adapt, in-context robot learning, foundation models for physical intelligence, and the evaluation of generalization. Its contribution is a way to measure and reward in-context adaptation, not a new learning paradigm.

Learning to adapt. Meta-learning trains a model to adapt from little data, either through a few gradient steps (Finn et al., 2017a) or entirely in its activations, with only the slow outer loop shaping the weights (Duan et al., 2016). Language models showed the second route at scale: a task specified only by examples in the prompt, with no gradient update (Brown et al., 2020). Algorithm Distillation (Laskin et al., 2023) and AdA (Adaptive Agent Team, 2023) carried in-context adaptation into reinforcement learning. Test-time training (Sun et al., 2020) instead updates parameters on each test input; Robotensor Competition’s setting rules that out, because weights are frozen at commitment.

In-context robot learning. One-shot imitation conditions a policy on one demonstration of a task it was not trained on (Duan et al., 2017; Finn et al., 2017b; Dasari & Gupta, 2020). ICRT

(Fu et al., 2025) prompts a causal transformer with teleoperated demonstrations and performs new tasks without updating its parameters. KAT (Di Palo & Johns, 2024) and RoboPrompt (Yin et al., 2025) get text-pretrained language models to imitate in context from keypoint or pose tokens. The Behavior Prompting Policy (Patel et al., 2026) treats a single demonstration, with observations downsampled and actions kept at full rate, as a prompt that programs a diffusion policy without gradient updates; Vector’s policy builds on it. Vector belongs to this line: the context is one robot demonstration with its actions.

From watching to acting. Policies conditioned on video learn from people as well as robots: BC-Z (Jang et al., 2021), Vid2Robot (Jain et al., 2024) and MimicPlay (Wang et al., 2023). Video generation has been used as a planner, with actions read from the generated future (Du et al., 2023; Wu et al., 2024). World Action Models predict future world states and actions together (Ye et al., 2026), and Zero-WAM (Zhou et al., 2026) follows a human video in context to perform tasks it never saw in training, including unseen RoboTwin 2.0 tasks. Horizon targets this line.

Foundation models and the turn to adaptation. Generalist policies trained on large multi-robot datasets push breadth: RT-1 and RT-2 (Brohan et al., 2023a; 2023b), Open X-Embodiment (Open X-Embodiment Collaboration, 2024), Octo (Octo Model Team, 2024), OpenVLA (Kim et al., 2024), π_0 (Black et al., 2025a) and $\pi_{0.5}$ (Black et al., 2025b). Industry now frames the frontier as adaptation. Skild AI describes an omni-bodied robot brain (Skild AI, 2025) and a model “built from the ground up as an in-context learner” (Skild AI, 2026). Generalist AI reports embodied foundation models that scale with physical interaction (Generalist AI, 2025) and then learn new tasks from one demonstration without gradient updates (Generalist AI, 2026). Each lab reports results on its own tasks. Robotensor Competition adds a common, open and adversarial measurement of the same capability.

Building blocks. `vector_v1.1` is assembled from standard parts: a ViT image encoder (Dosovitskiy et al., 2021) with CLIP’s architecture (Radford et al., 2021), a transformer decoder, and a 1-D temporal U-Net (Ronneberger et al., 2015) conditioned by FiLM (Perez et al., 2018), trained as a denoising diffusion model of action sequences (Ho et al., 2020; Janner et al., 2022; Chi et al., 2023) and sampled with DDIM (Song et al., 2021).

Evaluating generalization. RLBench (James et al., 2020), Meta-World (Yu et al., 2019), CALVIN (Mees et al., 2022), LIBERO (Liu et al., 2023), RoboCasa (Nasiriany et al., 2024), BEHAVIOR-1K (Li et al., 2024) and RoboTwin 2.0 (Chen et al., 2026) fix a task suite and an evaluation set in advance. They randomize the starting state far more than the task, so most measure state-space generalization. Generators such as GenSim (Wang et al., 2024) produce new tasks for training, not for evaluation after a model is frozen. Vector draws fresh evaluation scenes for every comparison from a chain block that did not exist when the entrant committed, compares two models on the same scenes, and sets task generation after commitment as its open problem.

Head-to-head and dynamic evaluation. Dynabench (Kiela et al., 2021) keeps benchmarks moving by collecting new examples against current models, and Chatbot Arena (Chiang et al., 2024) ranks models from pairwise human votes. Vector shares the pairwise idea, but its judge is a simulator, so both models face identical scenes and every comparison can be checked.

Incentive markets for models. Bittensor (Rao, n.d.) pays the miners its validators weight most. King-of-the-hill subnets for model training, such as Teutonic on subnet 3 (Teutonic, 2026), queue

challengers against a reigning model and pay the reigning line. Robotensor Competition applies that pattern to robot policies and adds chain-seeded, paired evaluation of in-context adaptation.

3. Problem setting: adaptive robot intelligence

An adaptive robot model is judged by what it does after it is shown the work, with its weights frozen. Robotensor’s competitions estimate that quantity, and each one fixes which parts of the work are new at test time and what the model is shown.

Tasks, contexts and frozen weights

Tasks. A task is a tuple $\tau = (E, O, N, G)$: an environment, the types of objects in it, their counts, and a goal program. An episode of τ starts from a state s drawn from $p(\cdot \mid \tau)$.

Context. The context c is everything the model is shown about the work besides its live observations: one robot demonstration with its actions (Vector), a video of a person or a robot doing the task (Horizon), or the model’s own experience with its outcomes.

Adaptive policy. A policy with weights θ acts from its observations so far and the context:

$$a_t \sim \pi_\theta(\cdot \mid o_{\leq t}, c) \quad (1)$$

θ is fixed from the moment it is committed on chain. Whatever the policy works out about the task, it works out in its context and its own state, and both are cleared between units. $Y(\theta; \tau, c, s) \in \{0, 1\}$ records whether the episode succeeds.

Objective. In Vector the context is the expert’s demonstration $c(s^d)$ from a scene s^d , and the policy is scored from a second, independent scene s . The quantity a competition maximizes is the mean over tasks of the expected success:

$$J(\theta) = \frac{1}{|\mathcal{T}|} \sum_{\tau \in \mathcal{T}} \mathbb{E}_{s^d, s \sim p(\cdot \mid \tau)} [Y(\theta; \tau, c(s^d), s)] \quad (2)$$

Two axes of adaptation

A test demands adaptation along two axes: what is new at test time, and what the context is. Each competition is a point on this map, and the roadmap is a path across it.

Everything below the top row is state-space generalization: the task tuple stays fixed and only its starting state or appearance changes. The top row is task-space generalization, a test task whose tuple appears nowhere in training. A new pose or a new texture is not a new task.

Recognizing a task versus learning one

In-context learning splits into two abilities (Pan et al., 2023). Task recognition infers which known task is wanted and applies what training taught about it; task learning acquires a mapping training never contained. Only task learning should keep improving with more or richer context.

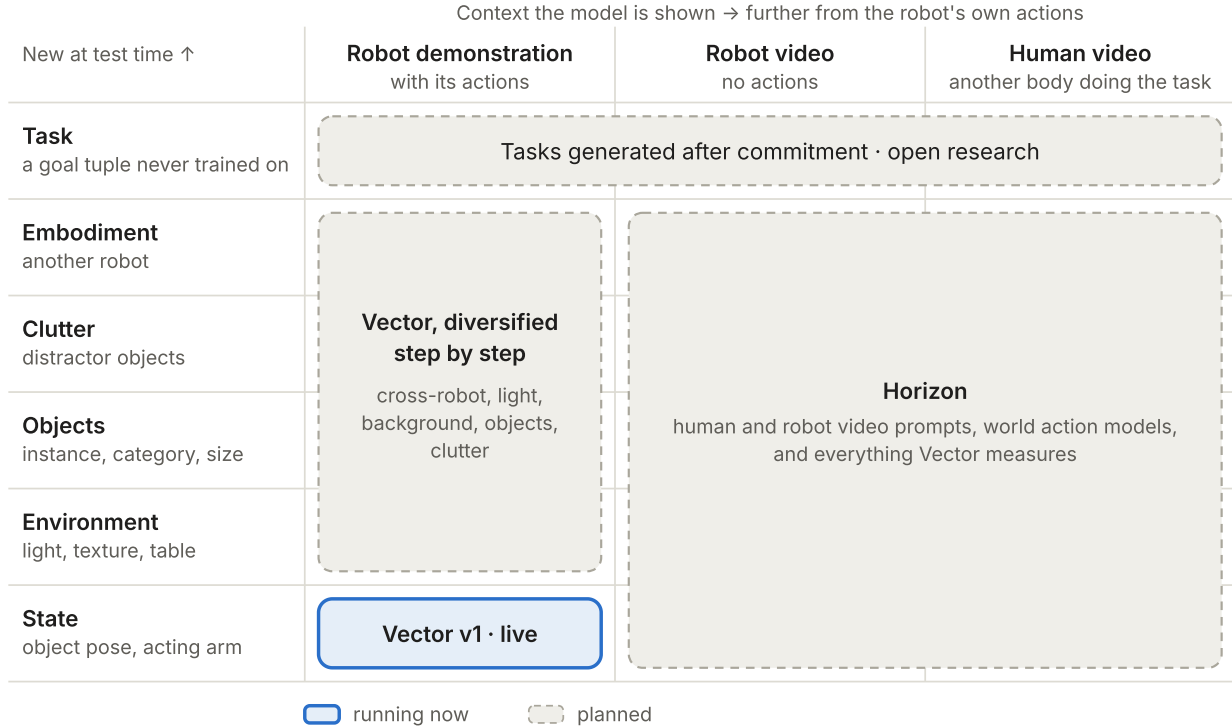


Figure 1. Robotensor’s competitions on the two axes of adaptation: what is new at test time (rows) against the context the model is shown (columns). Vector starts where adaptation is simplest; the roadmap climbs both axes.

Vector’s 16 tasks are public, and miners train on them. Vector v1 therefore rewards recognition plus execution under changed conditions, which is the skill a deployed robot needs when known work moves or varies. Task learning is measured only when the test task itself is new: held-out tasks, or tasks generated after the commitment (section 11).

Does the model use its context?

A policy can score well without reading its context if the scene alone gives the task away: a hammer and a block on the table already suggest what to do. The adaptation gain separates the two cases:

$$A(\theta) = J(\theta) - \tilde{J}(\theta) \quad (3)$$

$\tilde{J}$ is the score when each unit’s demonstration is replaced by a demonstration of a different task. A model that ignores its context loses nothing and has $A \approx 0$; a model that follows its context is misled and loses. Vector withholds the task’s name, the scene seed and the success condition, so the demonstration is the only explicit statement of the task. The network will publish A for every king (section 10).

What a measurement of adaptation requires

Table 1. What a measurement of adaptation requires, and how Vector v1 meets it.

Requirement	How Vector meets it	What remains open
Frozen weights	Weights are committed on chain with their sha256 before the duel and served by the validator’s own runtime	—
Context and test the model has not seen	Both scenes are drawn from a block finalized after the commitment, above the seeds reserved for training and evaluation	The scene distribution itself is public (section 11)
A context the model needs	No task name, seed or success condition; the scoring scene differs, so replay fails	The scene can hint at the task; measured by $A(\theta)$
The same test for both models	Both sides play the same units from the same demonstrations	—
Reproducibility	Each unit’s candidate scenes are a pure function of published values	The expert’s trajectory is not reproducible from its seed

A duel as an estimate

A duel. A duel between a king θ_k and a challenger θ_c draws $n = 10$ units per task, each a pair (s^d, s) , from a seed derived from the duel’s identity and a block hash. Both sides play every unit, so the estimate of $J(\theta_c) - J(\theta_k)$ is paired:

$$\hat{\Delta} = \frac{1}{|\mathcal{T}|} \sum_{\tau} \frac{1}{n} \sum_{i=1}^n (Y_{\tau i}(\theta_c) - Y_{\tau i}(\theta_k)) \quad (4)$$

The crown rule turns this estimate into a decision: the challenger is crowned when $\hat{\Delta} \geq 0.03$, a lead of at least 3 points on paired units. The rule is deliberately simple and transparent for Vector v1; as duel throughput grows, it tightens toward a statistical test on more units (section 11).

Constraints. Every entry is a point in Θ , the set of valid weight files: the same 753 tensors with the same shapes and types, loaded by the validator’s code. The policy sees nothing the robot could not observe. Each call must answer within the action budget, and the units of a duel are unknown to everyone until the seed block is finalized.

4. Vector: action-aware in-context adaptation

Vector is the first point on the map of section 3: the context is a robot demonstration with its actions, and what is new at test time is the starting state. A policy is shown an expert perform a task once in one scene, then must perform the same task, on its own, in a scene where the objects have moved and the acting arm may differ.

The task

Each evaluation unit pairs two scenes of the same task. The benchmark’s expert records a demonstration in the first; the policy receives that demonstration as its prompt and is scored on whether it completes the task in the second. The second scene moves the objects and may change which arm should act, so copying the demonstration’s motions does not succeed. Figure 2 shows the 16 tasks.

Duels run on [RoboTwin-Vector](#), Robotensor’s fork of the RoboTwin 2.0 bimanual simulator with a controlled scene-variation gate and a dedicated evaluation harness.

Table 2. Vector v1 benchmark parameters.

Parameter	Value
Robot	Two Franka arms
Difficulty	Vector v1: object position and, in most tasks, yaw vary, and so does which arm does the job when either can; lighting, background and object appearance are held fixed
Tasks	16: <code>beat_block_hammer</code> , <code>move_stapler_pad</code> , <code>place_bread_skillet</code> , <code>place_burger_fries</code> , <code>place_can_basket</code> , <code>place_cans_plasticbox</code> , <code>place_container_plate</code> , <code>place_dual_shoes</code> , <code>place_empty_cup</code> , <code>place_fan</code> , <code>place_mouse_pad</code> , <code>place_object_basket</code> , <code>place_object_scale</code> , <code>place_object_stand</code> , <code>place_phone_stand</code> , <code>place_shoe</code>
Units per duel	160 (10 per task)
Rollout cap	2× the expert’s steps in the unit’s scoring scene, within the task’s own step limit
Duel scene seeds	$[2,000,000, 2^{31} - 1)$, above the range $[0, 2,000,000)$ reserved for training and evaluation

Both scenes of a unit are made on demand and checked by the simulator’s scripted expert: it must solve each one, and the demonstration is recorded from its run in the first. A scene the expert cannot solve is rejected and replaced, never counted against a policy. Because duel seeds sit above the range reserved for training and evaluation, and are unknown until the seed block is final, no policy can have trained on a duel’s scenes.

The simulator runs from its own checkout and interpreter, and the policy runs in a separate process behind an authenticated socket, so neither imports the other’s code, and only arrays and JSON cross between them.

Design rationale

Each parameter of the benchmark isolates the skill Vector rewards, reading a demonstration and carrying it into a new layout, and keeps everything else out of the score.

The seed ranges guard against memorizing scenes, not against training on the scene distribution: anyone can generate unlimited data from the public simulator. Vector therefore measures generalization to new layouts within one distribution, which section 11 lists as a limitation.

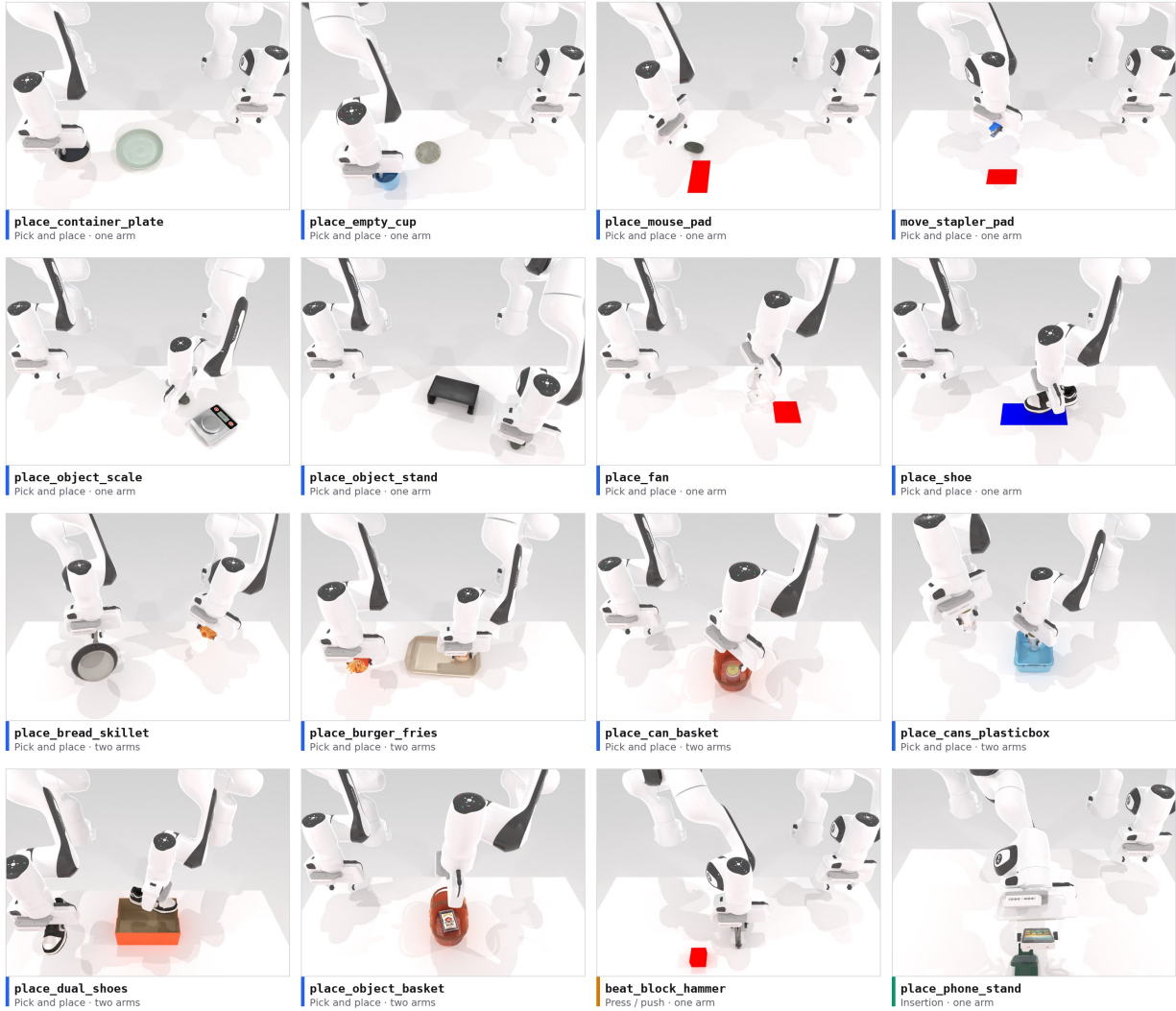


Figure 2. The 16 Vector v1 tasks, each shown mid-demonstration by the benchmark’s scripted expert on two Franka arms, in the Vector v1 setting (spatial variation only) and on scenes from the duel seed range. The bar beside each name marks its skill category: pick and place (blue), press/push (orange) and insertion (green).

Progressive diversification

Vector widens its scene distribution one factor at a time, so that every change in a king’s score can be attributed to a single source of variation. Vector v1 varies only where the task’s objects are and which arm acts; each later release switches on one more factor.

The scene generator draws random values in the same order whatever is switched on, and only overrides the factors a release does not vary, so a seed keeps its object layout as lighting and background are added. A king can then be replayed on its own units with one more factor switched on, and the drop in its score estimates the price of that factor. Published with every crowning once its replay tooling ships, this **adaptation profile** will show where a champion’s adaptation breaks, not only how well it does on the distribution it won on.

Table 3. Design rationale of the benchmark.

Choice	Value	Why
Difficulty	Spatial variation first	Vector v1 varies only object pose and which arm acts. Lighting, background, object size, colour and category, and clutter are introduced one at a time in later releases, so each change in a king’s score has a single cause.
Nested variation	A seed keeps its layout as scene factors are added	The gate draws the scene’s random values in the same order whatever is switched on and overrides the factors a release does not vary, so a seed keeps its object layout when lighting, background or table height are switched on. Object-category variation is held to the same property before it ships.
Cross-scene units	Demonstration and scoring scene drawn independently	Replaying the demonstration’s motions fails when the objects have moved, so a policy must use the demonstration as a description of the task, not as a trajectory.
Step cap	2× the expert’s steps in the unit’s scoring scene	Ties the time allowed to the unit’s own difficulty instead of a fixed horizon, and bounds the length of a duel.
Demonstration filter	Kept only if the expert succeeds	A unit without a valid demonstration says nothing about a policy. A scene whose objects do not settle, or in which the expert cannot plan, raises an error, fails or exceeds a joint limit, is rejected, and the next of the unit’s 20 candidate seeds is tried. A unit is void only if all 20 fail.
Void limit	20% of units	A harness fault is excluded from both sides, which keeps the comparison paired; past 20%, too much of the sample is missing for the result to stand.
Seed ranges	Duels draw from $[2,000,000, 2^{31} - 1)$	The benchmark reserves seeds below 2,000,000 for training and evaluation, and a duel’s seeds are unknown until its seed block is final, so no policy can have trained on a duel’s exact scenes.
Runs per task	10 today (160 units)	Sets the duel’s cost and its statistical resolution; it rises as validator throughput allows.

Clutter matters for a second reason. A scene that holds only the task’s own objects can give the task away; distractors make the demonstration necessary to know what to do, which raises the adaptation gain $A(\theta)$ a release can demand.

Task and skill expansion

Vector v1 concentrates on one skill so that the adaptation it measures is not confounded with skill coverage: 14 of its 16 tasks are pick-and-place, with one press/push task and one insertion task, and 10 of the 16 are performed by one arm while 6 need both. The task set then grows by skill category, one category at a time:

Each expansion adds a category whose tasks the expert solves reliably, reports the score per category as well as overall, and is published as a new contract version. A duel’s id includes the contract version, so results under different task sets are never mixed. Expansion also changes what the benchmark measures. With more tasks sharing the same objects and scenes, the demonstration

Table 4. Variation axes, introduced one factor at a time.

Variation axis	What varies	What it tests	Status
Spatial	Object position and, in most tasks, yaw; the acting arm	Mapping a demonstrated behaviour onto a new layout and, where needed, onto the other arm	Live
Lighting and background	Light sources, wall and table texture, table height	Visual invariance of the encoder and of the prompt’s reading of the scene	Next
Object appearance	Object size and colour	Whether the policy keys on the object’s role rather than its pixels	Planned
Object category	Other instances and alternative categories of the task’s objects	Whether a demonstration with one object transfers to another that plays the same role	Planned
Clutter	Distractor objects on the table	Attention to the task-relevant objects, and dependence on the demonstration to identify them	Planned
Embodiment	A robot other than the one in the demonstration	Transfer of demonstrated behaviour across bodies	Planned

Table 5. Skill categories of the task set.

Skill category	In Vector	Examples
	v1	
Pick and place	14	place_container_plate, move_stapler_pad, place_dual_shoes
Press / push	1	beat_block_hammer, press_stapler, turn_switch
Insertion	1	place_phone_stand, hanging_mug
Stacking	0	stack_blocks_three, stack_bowls_two, blocks_ranking_size
Open / close	0	open_laptop, open_microwave, put_object_cabinet
Bimanual coordination	0	handover_block, lift_pot, grab_roller
Articulated manipulation	0	rotate_qrcode, shake_bottle, adjust_bottle

becomes the only way to tell them apart; and a category held out from training, introduced only at evaluation, is the first step from task recognition toward task learning.

5. The **vector_v1.1** policy

vector_v1.1 is an in-context learner built on the Behavior Prompting Policy (BPP) architecture (Patel et al., 2026), configured for bimanual manipulation in RoboTwin and trained from scratch. The demonstration is encoded once into a prompt memory, and every prediction reads the live frames against it. The context enters only through the forward pass; nothing in the weights changes at test time. Every miner trains the same network, and only the weights differ.

Lineage. BPP introduced the design **vector_v1.1** keeps: a behavior prompt in which observations are downsampled while actions stay at full rate; per-chunk attention pooling into one prompt token; a six-layer transformer prompt encoder (width 768, MLP 3,072, 8 heads); learned attention sinks;

and a CNN U-Net diffusion decoder with FiLM conditioning (channels 256–512–1,024, kernel 5), with the demonstration encoded once rather than at every denoising step. BPP’s published results support the design: an 80.7% error reduction over goal-image conditioning and 33.3% over ICRT on unseen drawing tasks, and on unseen LIBERO-Gen task combinations results that rival $\pi_{0.5}$ fine-tuned for 100,000 LoRA steps, without pretraining. **vector_v1.1** fixes BPP’s configuration for RoboTwin’s two Franka arms: three camera encoders for the head and both wrists, six per-key pose tokens, a bimanual end-effector action space with 6-D rotation, a two-frame history, and inference pinned at 16 DDIM steps.

Table 6. The **vector_v1.1** network at a glance.

Component	Design
Observation	9 tokens per frame: three camera embeddings (head and both wrists, one CLIP ViT-B/16 encoder per camera at 224 px) and six proprioception projections
Demonstration prompt	One frame per 15 steps; each frame’s 9 tokens and its 15 actions pooled into a single token
Context	The two most recent frames cross-attend the prompt in a 6-layer transformer decoder (width 768)
Action head	A 1-D U-Net that denoises 16 future actions in 16 DDIM steps; 12 are executed before the next prediction
Size	753 tensors, about 781M parameters

The tensor layout is pinned by the validator and published with the runtime, so a miner knows exactly what a valid file contains before training.

Architecture in detail

The demonstration is encoded once per unit into a prompt memory; every prediction, one per 12 control steps, then reads the two latest frames against that memory and denoises a 16-action chunk, of which 12 are executed.

Design decisions

Each component of **vector_v1.1** answers a specific requirement of in-context control. Table 7 gives, for each decision, the main alternative it was weighed against, why the choice was made (citing BPP’s ablations where they bear on it), and the cost it accepts.

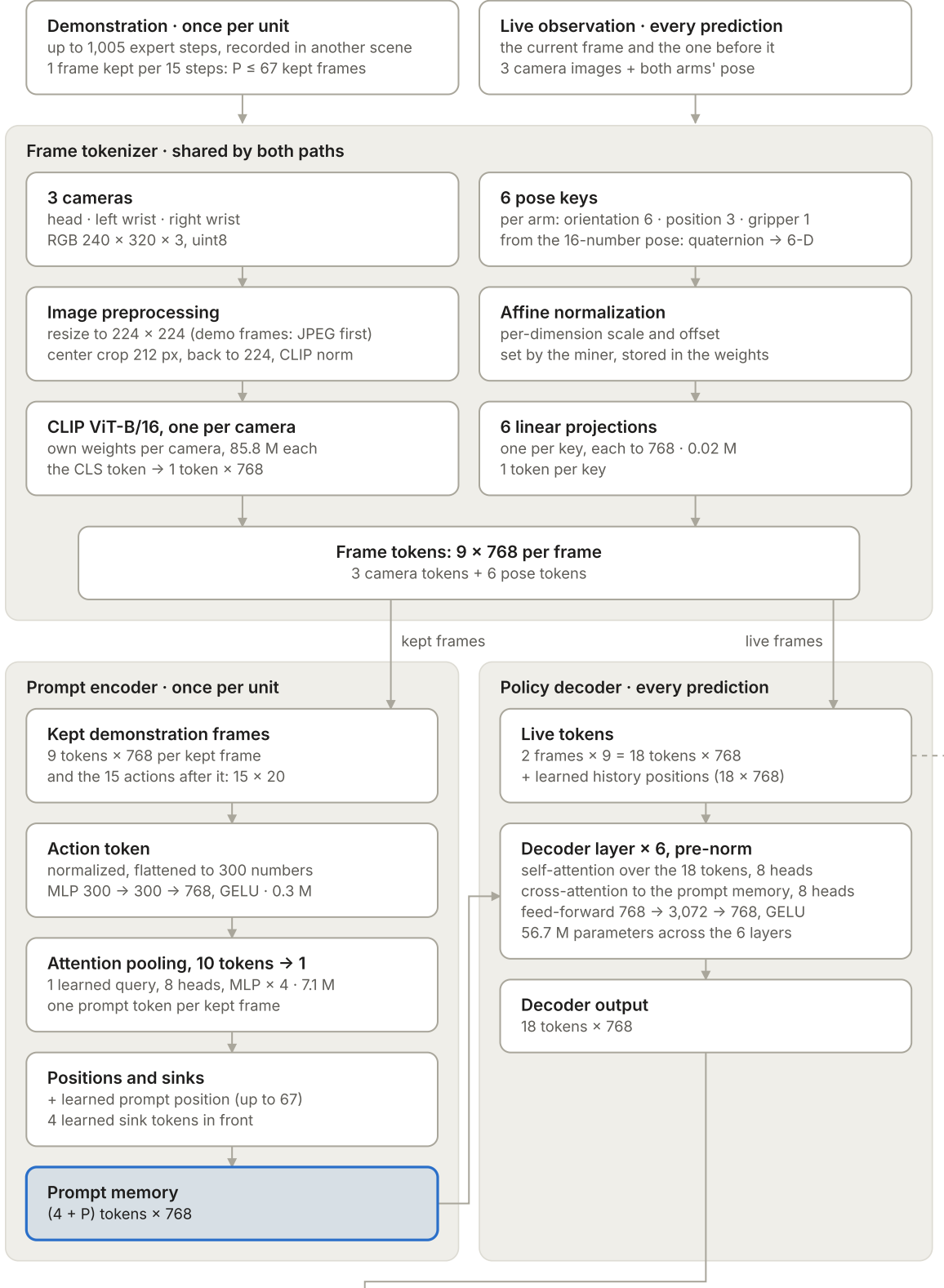


Figure 3. The **vector_v1.1** architecture: tokenizer, prompt encoder, policy decoder and diffusion head (781.0M parameters in 753 float32 tensors; width 768; 8-head attention throughout). The network reads one demonstration and two live frames and returns 16 actions. Part 1: tokenizer, prompt encoder and policy decoder.

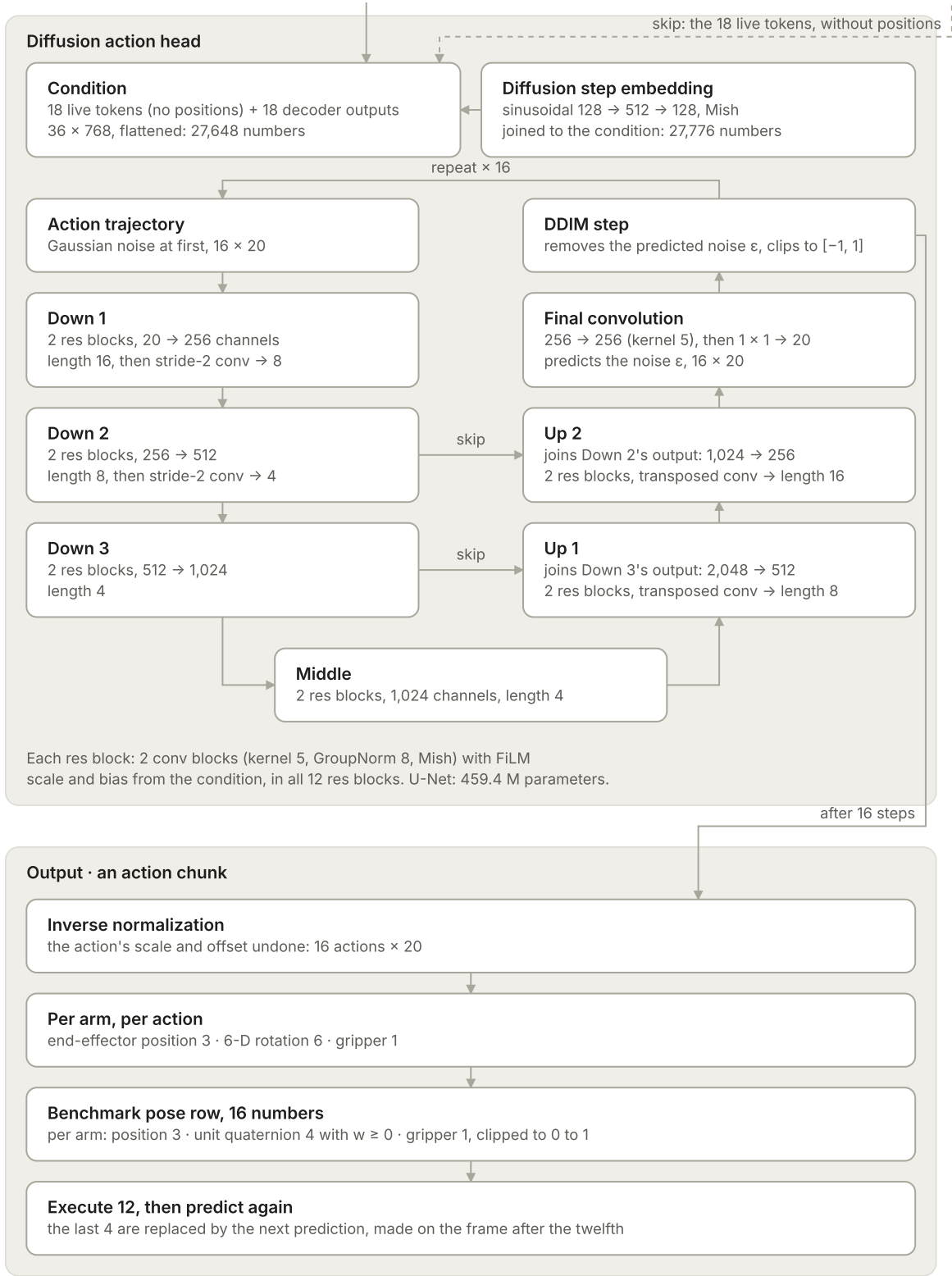


Figure 3. (continued) Part 2: diffusion action head and output.

Table 7. Design decisions in `vector_v1.1`.

Decision	Choice in <code>vector_v1.1</code>	Main alternative	Why this choice	Cost accepted
Visual encoder	CLIP ViT-B/16 architecture, one per camera, CLS token	One shared encoder; patch tokens	Head and wrist cameras see different statistics (scale, motion, occlusion); one token per camera keeps a frame at 9 tokens	257.4M parameters, 171.6M more than one shared encoder; spatial detail compressed to one token per view
Proprioception	One linear token per pose key, six in all	One concatenated state token	Each arm’s orientation, position and gripper can be attended separately, which matters when the acting arm changes	Six tokens instead of one
Prompt granularity	One frame per 15 steps, with the 15 actions after it	Every frame	BPP’s ablations: observations are necessary in the prompt, actions carry the transitions between them, and downsampling observations below about 1 Hz hurts. At most 71 memory tokens instead of 674	Visual detail between kept frames
Prompt compression	Attention pooling of a chunk’s 10 tokens (9 for the frame, 1 for its 15 actions) into one, with a learned query (Lee et al., 2019)	Keep all tokens; mean pooling	BPP shows per-chunk pooling shortens the prompt and ties its modalities together in time; up to 9.5× fewer keys in every cross-attention	7.1M parameters; a per-frame bottleneck
Attention sinks	Four learned tokens before the prompt (Xiao et al., 2024)	None	Used by BPP for LIBERO-Gen; cross-attention has somewhere to put weight when no prompt frame is relevant	Four tokens
Observation history	Two frames	One frame; four or more	Two frames carry velocity; each extra frame adds 9 decoder tokens and 13,824 numbers to the condition	No long-range memory within an episode
Context reasoning	Six pre-norm decoder layers, self- then cross-attention, width 768, 8 heads (Xiong et al., 2020)	Four or eight layers; one encoder over prompt and observation together	Cross-attention reads the prompt as memory without lengthening the sequence; pre-norm trains stably at depth	56.7M parameters

Decision	Choice in <code>vector_v1.1</code>	Main alternative	Why this choice	Cost accepted
Condition for the action head	The 18 frame tokens and the decoder’s 18 outputs, flattened: 27,648 numbers, plus 128 for the step	Pooled vector; cross-attention inside the U-Net	No bottleneck between context reasoning and action generation	398.2M parameters of FiLM projections (below)
Action generator	1-D temporal U-Net trained as a denoising diffusion model, ϵ -prediction, 16 DDIM steps (Janner et al., 2022; Chi et al., 2023)	Direct regression; flow matching (Lipman et al., 2023); discrete action tokens	Represents several valid ways to do a task (which arm, which grasp) without averaging them	16 U-Net passes per prediction
Action space	Per arm: end-effector position, 6-D rotation and gripper; 20 numbers (Zhou et al., 2019)	Joint positions; quaternions	A continuous rotation representation, and an action space shared by different arms, the basis for cross-robot transfer	Depends on the controller’s inverse kinematics
Action chunking	Predict 16 actions, execute 12 (Zhao et al., 2023)	Execute all 16; execute 8	Smooth, temporally consistent motion, re-planned every 12 steps from fresh observations	One prediction per 12 control steps
Numeric format and size	781M parameters, float32	bfloat16; a smaller network	One numeric format for every entry, so the header check fully specifies a valid file; 3.1 GB on disk	Memory and bandwidth

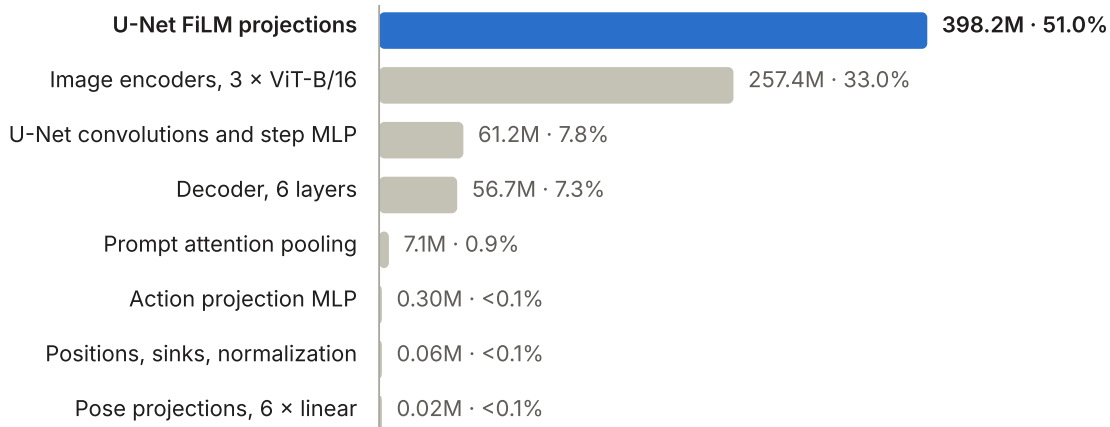


Figure 4. Parameters of `vector_v1.1` by component, in millions (781.0M in total), exact counts from the architecture definition. Half of the parameters condition the action head.

Where the parameters go

Half of the network’s parameters sit in one place: the FiLM projections that inject the condition into the action head.

Each of the U-Net’s 12 residual blocks projects the full 27,776-number condition to a scale and a bias for its channels, and those projections hold 398.2M parameters, against 61.0M for all of the U-Net’s convolutions. It is the price of conditioning without a bottleneck. Each projection applies Mish element-wise and then a linear map, and only the 128 step numbers change between denoising steps, so the observation part of each projection can be computed once per prediction instead of 16 times, with mathematically identical outputs.

Starting point and training scale

The throne starts empty: the first admitted entry takes it by genesis, scored on its own units, and every later king must beat the reigning one; after a king is vacated, the next entry again takes the empty throne by genesis. What the architecture fixes for every entry:

- **Noise schedule.** A 50-step squared-cosine noise schedule, ϵ -prediction, and predicted samples clipped to $[-1, 1]$ in normalized action space; inference runs 16 DDIM steps ($t = 45, 42, \dots, 0$).
- **Normalization.** Every pose key and the action carry a per-dimension scale and offset inside the weights, set by whoever trains them; the validator applies them as stored. The action normalization should map actions into $[-1, 1]$, since samples outside it are clipped.
- **Inputs.** Demonstration frames go through a JPEG round trip and are resized to 224×224 , as recorded episodes are; live frames skip the JPEG step. Every frame is then center-cropped to 95% (212 px) and resized back to 224 before the encoder.

BPP reports training runs of 27 to 42 hours on four GPUs per setting, a useful guide to the scale of a competitive first entry.

Why one fixed architecture, and why this size

Fixing the network trades architectural freedom for three things an open competition cannot do without: a fair comparison, a safe validator, and a uniform cost per duel.

- **Fairness.** When every entry fills the same 753 tensors, a duel compares training, data and method. A larger or slower model cannot win by spending more compute at inference.
- **Safety.** A submission is data, never code. The validator reads the safetensors header, checks every name, shape and type against the pinned manifest before reading a tensor, builds the network with its own code and loads the weights into it. Nothing a miner uploads is executed or unpickled.
- **Uniform cost.** Every entry does the same work per prediction, so the time budgets bind the harness, not the model: 60 s per action call, 600 s to start a policy, 1,200 s of policy time per unit, and at most 24 hours of work per duel. A duel’s cost is known before it starts.
- **Size.** At 781M float32 parameters the weights are about 3.1 GB. One unit, simulator and policy together, used up to 34 GB of GPU memory in our measurements, so an 80 GB card can run two units side by side; units per card is a validator setting.

The cost is that miners cannot innovate on the architecture itself. The architecture is versioned instead: the spec pins its name and the digest of its tensor manifest, and a new version (`vector_v1.2`, say) would be announced in advance and introduced as a new spec, never changed under running duels.

Adaptive model training

A miner does not train a policy for sixteen tasks; it trains a learner that turns one demonstration into the right behaviour in a scene it has not seen. With the network and its runtime fixed, everything that decides how well that learner adapts is left to the miner:

Because the architecture, the runtime and the units are the same for both sides, a duel between two entries is a controlled experiment on these choices. The winner’s weights are public from its commitment onward, and the record says exactly what it beat and by how much, so every crowning is a public data point on which adaptive training works.

6. The competition mechanism

A miner enters by committing weights on chain; validators settle every entry in a king-of-the-hill duel against the current best model.

Submission and commitment

A submission is a Hugging Face model repository holding one file, `model.safetensors` (stored with Git LFS), plus optionally `README.md` and `.gitattributes`. Nothing else is accepted, and nothing a miner uploads is ever executed or unpickled.

A miner claims a submission by writing a commitment to the chain from its hotkey:

Table 8. What miners control in adaptive model training.

Lever	What it decides
Context pairing	Which demonstration prompts which target episode. Pairs must differ the way a duel’s do (same task, another scene, possibly another arm) or the model learns to replay.
Context-dependence training	Whether some pairs carry a demonstration of the wrong task, so the model learns to follow its context rather than guess the task from the scene. This is what raises the adaptation gain $A(\theta)$.
Data	How many tasks and episodes, drawn from the public simulator with the miner’s own data pipeline (BPP’s ablations favour many tasks with few demonstrations each over few tasks with many); whether to train under wider variation than the current release, so the model is robust before the benchmark diversifies.
Curriculum	The order of difficulty: same scene before cross-scene, easy tasks before hard ones, narrow variation before wide.
Objective and augmentation	The denoising loss on action chunks, image augmentation, and the normalization statistics stored in the weights.
Compute	Model updates, batch size and schedule within the miner’s budget.

```
vector:<owner>/<repo>@<commit>.<digest>
```

The commitment pins the repository’s exact revision (the 20-byte commit sha) and the sha256 of `model.safetensors`, encoded in base64url. Weights that do not hash to the committed digest are refused.

The recommended order protects the miner’s work: upload to a private repository, commit on chain, then make the repository public immediately. The chain timestamps the claim before anyone can see the weights, and of two commitments carrying byte-identical weights, the first admitted keeps them. The queue runs oldest first, so that is the earlier commitment, provided its repository is public when its turn comes.

One submission per hotkey. The first commitment of a hotkey to reach the queue is its only one. Anything the same hotkey commits afterwards is refused, whatever became of the first.

King-of-the-hill duels

The validator serves its queue oldest commitment first, and each queued submission duels the reigning king once; the crown moves only when the challenger leads the king by the crown margin.

1. **Queue.** A commitment is queued the moment the validator reads it from the chain.
2. **Seed.** When the duel starts, the validator takes the chain’s finalized head, which must come after the challenger’s commitment. Its hash, with the duel’s id, deals the 160 units.
3. **Check.** The challenger and the king are looked up on the Hub and their headers checked against the pinned architecture.

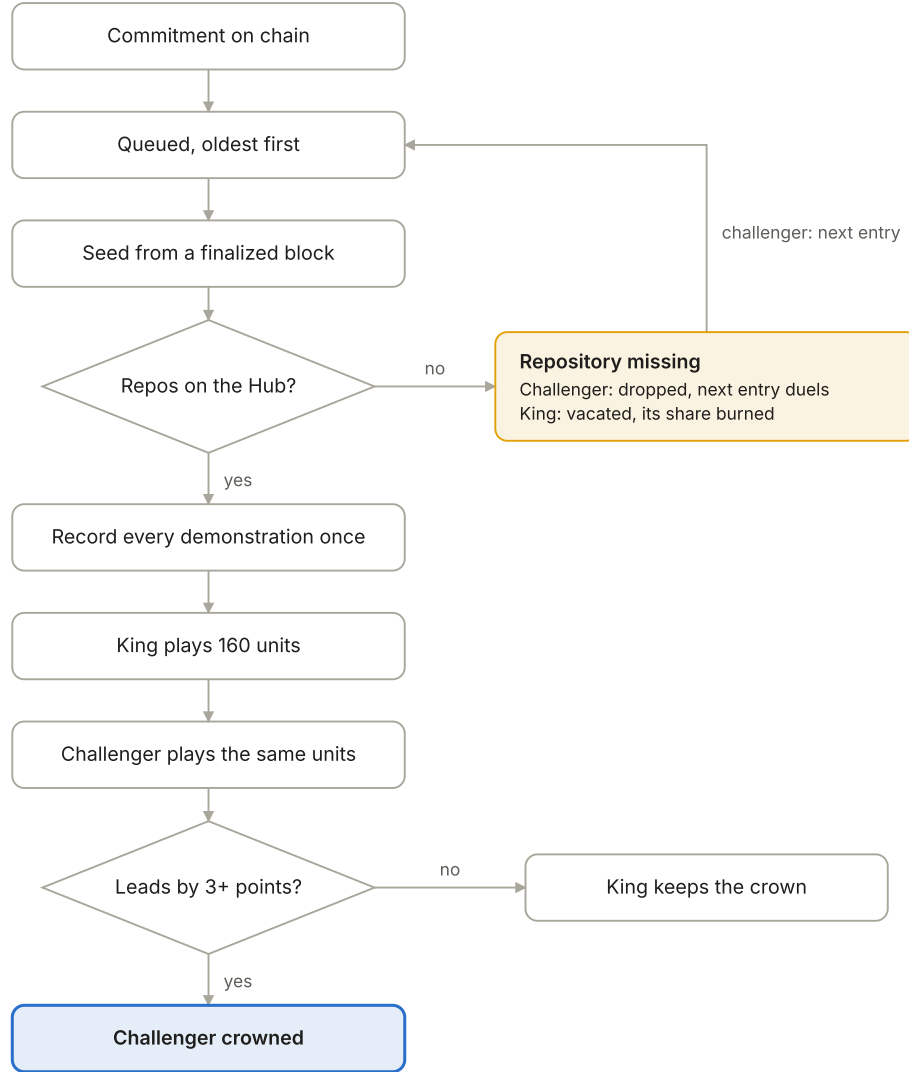


Figure 5. One duel, from commitment to crown. A challenger is crowned only when it leads the king by 3 points.

4. **Demonstrations.** Every unit’s demonstration is recorded once, and both sides are prompted with exactly the same ones.
5. **Play.** The king plays every unit, then the challenger plays the same units.
6. **Score.** A side’s score is its success rate per task over the units that are not void, averaged over the tasks that have at least one such unit.
7. **Verdict.** The challenger is crowned when its average beats the king’s by at least 3 points; otherwise the king keeps the crown. The record is published either way.

A missing repository never stalls the queue: a missing challenger is dropped for the next entry, a missing king is vacated.

A unit is void only for a harness cause, such as a simulator fault or the duel’s time budget running out, never for what a policy does; a unit void on either side is void for both. If more than 20% of a

duel’s units are void, the whole duel is void: no record is published, the crown stays put, and the challenger’s submission is spent. With no king at all, the first entry takes the throne by **genesis**, scored on its own units.

7. Threat model and defenses

Every rule below exists so that the only way to win is to train a better policy.

Table 9. Integrity rules applied by validators.

Situation	What the validator does
Challenger’s repository missing at its turn (deleted, private, revision gone, or the Hub not answering)	Dropped as missing ; the next entry in the queue duels instead. The hotkey’s one submission is spent.
King’s repository missing when the validator checks it, before every queue step (including a Hub that does not answer)	The king is dethroned without a duel (a vacate record), and its champion place is kept but its emission is burned. The next entry takes the empty throne by genesis. If the repository disappears during a duel instead, the king forfeits: every king unit counts as a failure.
Repository holds files other than the weights, README.md and .gitattributes	Refused.
Weights do not match the committed digest	Refused.
Weights byte-identical to an earlier commitment’s	Refused as a duplicate; the first commitment admitted with them keeps them.
Same hotkey commits again	Refused: one submission per hotkey.

No submitted code runs. The validator reads the safetensors header and checks every tensor name, shape and type against the pinned architecture before reading any tensor. It then builds the network with its own code and loads the weights.

No one can tune for the test. Duel scenes are drawn from a finalized block that comes after the commitment, and from seeds above those reserved for training and evaluation. The validator reads the block and its hash from the chain itself, never from a miner.

Every duel can be checked. A duel’s id and its units’ candidate scene seeds are pure functions of published values, so anyone holding its record can deal the same units again and check the scoring scene each unit used.

Threat model

The adversary is any miner or validator who wants emission without producing a better policy. The rules above close the cheap attacks; the table gives each remaining threat’s residual risk and the measure against it.

Pricing the copy attack. A perturbed copy wins first place, 12% of miner emissions, with probability r per attempt, and holds it for one crowning interval on average. If a hotkey costs c

Table 10. Threat model.

Threat	Defense today	Residual risk	Recommendation
Resubmitting the king’s exact weights	Byte-identical weights belong to the first commitment admitted with them	None	—
Submitting a lightly perturbed copy of the king	One submission per hotkey; 3-point margin	A near-copy can clear the margin by chance	A near-duplicate check against recent champions; a tighter crown rule as throughput grows; registration cost above the expected gain
Copying a miner’s weights before they commit	Upload private, commit the digest, then publish	None once the original is admitted: a later identical commitment is a duplicate	—
Flooding the queue from many hotkeys	One submission per hotkey; oldest first	Each plausible entry costs a full duel and delays honest ones	Price registration above one duel’s validator cost; stop a duel early once its result is certain
Malformed weight files	Header size cap, manifest check before reading, finite values, non-zero scales, repository size cap	None known	—
Memorizing evaluation scenes	Duel seeds above the range reserved for training and evaluation; seed block finalized after the commitment	None	—
Training on the public simulator	Allowed, and the intended way to compete	Scores measure generalization within one scene distribution	Wider variation and new tasks in later releases (section 11)
Code in a submission	None runs: the validator’s own runtime serves the weights, in a separate process behind an authenticated socket	None	—

to register, copying pays whenever $r \times (\text{expected earnings of a crown}) > c$. A near-duplicate check and a tighter crown rule lower r ; the registration cost raises c .

Pricing queue spam. A weak entry costs nothing to the attacker beyond its registration, but a full duel to every validator, and it delays every honest entry behind it. Two levers bring that cost down: a registration price at least equal to a duel’s validator cost, and ending a duel as soon as its remaining units can no longer change the verdict, which leaves every verdict as it would have been.

8. Transparency and reproducibility

Every duel’s outcome is written to an append-only result store and mirrored to a public Hugging Face dataset, which the competition dashboard reads.

Table 11. The public result store.

File	Contents
index-NNNN.jsonl	One canonical record per event: a genesis, a duel or a vacate
events/<event_id>.json	The units, per-task scores, verdict and seed block of one event
media/	Clips of the demonstrations and of both sides’ rollouts, stored by content hash
head.json	The current king
queue.json	The submissions waiting to duel
running.json	The duel in progress and how far it has got
champions.json	The champions paid now and each one’s share

The index is the source of truth and only ever grows. Because units derive from the duel’s id and a published block hash, a third party can recompute any duel’s candidate scenes and check each published scoring scene against them.

9. Incentives and economics

Vector receives 30% of miner emissions, split among its four most recent champions 40% / 30% / 20% / 10%, newest first; everything else goes to the burn UID until further competitions launch.

Table 12. Champion places and their shares.

Place	Share of Vector’s 30%	Share of all miner emissions
1 (newest champion)	40%	12%
2	30%	9%
3	20%	6%
4	10%	3%

- **Fewer than four champions.** The filled places share the whole 30% in the same proportions.
- **Deregistered champion.** Its part is burned, not handed to the other champions: one champion’s reward never grows because another left.
- **Vacated king.** A king dethroned because its repository disappeared keeps its place in the ranking, and its part is burned.
- **Burn.** What no champion is paid goes to the subnet owner’s hotkey, the burn UID.

Weights are keyed by hotkey and mapped to UIDs only when set, because UIDs are recycled. Validators set weights every 360 blocks (one tempo, about 72 minutes), on a thread separate from the duels, so a duel lasting hours never delays a validator’s weights.

Economics

A champion’s expected lifetime earnings equal Vector’s share of miner emissions over one mean interval between crownings; the 40/30/20/10 split decides when that is paid, not how much.

Let E_m be miner emissions per unit time, $s = 0.3$ Vector’s share, $w = (0.4, 0.3, 0.2, 0.1)$ the split, and T_k the time a champion spends in place k , which is the interval between two later crownings. A champion that stays registered earns:

$$\mathbb{E}[\text{earnings}] = s E_m \sum_{k=1}^4 w_k \mathbb{E}[T_k] = s E_m \bar{T} \quad (5)$$

The last step holds when crownings arrive at a steady rate, with mean interval $\bar{T}$, because the parts of w sum to 1. The front-loaded split rewards a champion fastest while it is newest, and keeps paying three predecessors so that a narrow loss does not end a miner’s income at once.

Under dynamic TAO, miners receive 41% of a subnet’s emission, validators and their stakers 41%, and the subnet owner 18% ([Bittensor documentation](#), at the time of writing; the owner’s share is a governance parameter). Vector’s 30% of miner emissions is therefore about 12.3% of the subnet’s emission, and a first place about 4.9%.

A miner’s decision. An attempt costs a registration c and a training budget C . It pays when:

$$P(\text{crown}) \cdot s E_m \bar{T} > c + C \quad (6)$$

The crown rule enters through $P(\text{crown})$, which rises steeply with how much better the challenger truly is: the competition pays for real improvement.

A validator’s cost. Each duel costs G GPU-hours on the validator that plays it. A duel gets at most 24 hours of work, so a running validator settles at least one challenger a day; the queue’s real throughput is 24 h divided by the measured duel time.

The burn. At launch Vector pays 30% of miner emissions and the rest is burned, as are Vector’s own unpaid parts: places with no champion, deregistered champions and vacated kings. The burn falls as further competitions open; Horizon’s share is 0% until it does.

Table 13. Quantities in the economic model.

Quantity	Symbol	Value
Vector’s share of miner emissions	s	30%
Champion split, newest first	w	40 / 30 / 20 / 10%
Mean interval between crownings	$\bar{T}$	Set by the market; measured after launch
Registration cost	c	Set by the chain at the time of registration
GPU-hours per duel	G	To be measured

10. What the network publishes

From the first duel, every outcome is public, so the competition’s progress is measured in the open rather than reported after the fact.

Table 14. What the network publishes.

Published	From	What it shows
Every duel: units, per-task scores, verdict, seed block, clips	The first duel	Who beat whom, on which scenes, and why
The reigning king and the four paid champions	The first genesis	Where the frontier stands and who is paid
The queue and the duel in progress	Launch	When an entry will be played
Zero and replay baselines on the duel units	As the tooling ships	That copying the demonstration does not solve the benchmark
The adaptation gain $A(\theta)$ of each king	As the tooling ships	Whether the champions follow their demonstrations
The adaptation profile of each king	With each new variation factor	Where a champion’s adaptation breaks
Duel time	Launch	The queue’s throughput and a validator’s cost

The dashboard reads the same append-only store the validators write, so every number on it can be traced to a record and re-derived.

11. Limitations and roadmap

Status at launch

Table 15. What runs at launch and what follows.

Component	Status
Weights-only submissions with on-chain commitments	Live at launch
King-of-the-hill duels on 160 units seeded after the commitment	Live at launch
The 16 Vector v1 tasks with spatial variation	Live at launch
Missing-repository handling: dropped challengers, vacated kings	Live at launch
Champion pool (40 / 30 / 20 / 10%) and burn	Live at launch
Public result store and dashboard	Live at launch
Zero and replay baselines on duel units	Planned
Adaptation gain and adaptation profile for every king	Planned
Lighting, background, object appearance, object category, clutter	Planned, one factor at a time
New skill categories	Planned, one category at a time
Near-duplicate check and a tighter crown rule	Planned
Cross-robot evaluation	Planned
Robotensor Horizon	Planned
Tasks generated after commitment	Research

Roadmap

The roadmap is a path across the map of section 3: up the first column with Vector, along the context axis with Horizon, and finally to the top row, tasks that did not exist when the model was committed. Each competition runs beside the others with its own share of emissions.

- **Robotensor Horizon** moves along the context axis. A human or robot video and the robot’s history are the context, and the model predicts the future physical state together with the robot’s actions, joining human-video in-context learning, World Action Models and efficient inference in one competition. Its share stays at 0% until it opens.
- **Up the Vector ladder.** Lighting and background, object size, colour and category, and clutter are introduced one factor at a time on the same layouts; new skill categories join the task set; and cross-robot evaluation adds another embodiment (section 4). Each step makes the demonstration carry more of the task.
- **More of the competition in the open.** The result store, the dashboard and the evaluation code are already public; each new competition ships the same way.
- **Tasks generated after commitment.** The step from recognizing tasks to learning them. A task tuple (E, O, N, G) drawn from a chain block after the commitment needs a task compiler: semantic validation, a feasibility check, a success predicate, and a scene and demonstration that satisfy them. This is open research, and the protocol of section 6 is built to carry it.

- **Adaptation in every record.** The adaptation gain and the adaptation profile of every king, published beside its score, so that progress on adaptation is visible and not only progress on the score.
- **A tighter crown rule.** As duel throughput grows, the fixed margin moves to a statistical test on more units per duel, published as a new contract version.

Limitations

Vector measures a narrow, well-defined skill, and its first version leaves several questions open.

- **Simulation only.** No duel touches a real robot, so transfer to hardware is not measured.
- **One distribution.** Anyone can train on unlimited data from the public simulator. Vector v1 measures state-space generalization, new layouts within that distribution, without visual variation; later releases add it step by step.
- **Recognition, not yet learning.** The sixteen tasks are known to every miner, so Vector v1 rewards recognizing a task from its demonstration and executing it under change, not learning a task never seen. Task-space generalization waits for tasks generated after commitment.
- **One embodiment, sixteen tasks.** Two Franka arms and mostly short placement tasks. A champion here is not a generalist.
- **Scripted demonstrations.** Every prompt is a successful demonstration by the simulator’s expert. Human, imperfect or cross-embodiment demonstrations are not tested.
- **A fixed architecture.** Miners improve weights, not networks, and the 781M-parameter design sets the ceiling until a new version is specified.
- **A first-release crown rule.** The 3-point margin on 160 paired units is simple and transparent, and its resolution grows with the units a duel can afford.
- **One availability dependency.** Submissions live on Hugging Face, and an outage there counts as a missing repository.

12. Conclusion

Robotensor Competition treats adaptation as the product. A robot model is worth what it does after it is shown new work, with its weights frozen, and the competition measures exactly that: on scenes and demonstrations drawn after the model is committed, against the best model so far, with the result published for anyone to check.

Vector is the first point on a map with two axes, what is new at test time and what the model is shown, and the roadmap is a path across it toward tasks that did not exist when a model was committed. One fixed in-context architecture turns the competition into a market for adaptive model training, where every crowning is a public data point on which training produces a model that adapts.

Vector v1 is deliberately simple where simplicity buys transparency: one task family, spatial variation and a fixed crown margin. Each grows in announced, versioned steps, the benchmark toward wider variation and new skills and the crown rule toward a statistical test on more units. What

the network publishes, the adaptation gain above all, will show whether the champions adapt or merely recognize.

Disclaimer

This white paper describes the design of the Robotensor Competition subnet as of its version date and contains forward-looking statements about planned features, which may change or not be delivered. It is not an offer of securities or investment advice, and nothing in it is a promise of returns, emissions or token value. Participants are responsible for their own decisions and for compliance with the laws that apply to them.

Appendix A. Participating

Miners need a training setup and a Hugging Face account; validators need one GPU host that can run the simulator and the policy side by side.

Miners

1. Train `vector_v1.1` weights with your own data and method, and save them as `model.safetensors`.
2. Check the file’s header and tensors against the pinned architecture: `robotensor miner check --dir submission/`.
3. Submit: `robotensor miner submit` uploads to a new private repository (an existing repository keeps its visibility), commits on chain, then makes the repository public.
4. Keep the repository public; a reigning king whose repository disappears is dethroned and its share burned.

Validators

Table A1. Validator environments.

Environment	Python	Runs
Simulator	3.10	RoboTwin-Vector and the policy protocol
Policy	3.12	The validator’s own <code>vector_v1.1</code> runtime on torch 2.8 (CUDA 12.8)
Host	3.12	The subnet validator and the duel engine

One evaluation unit can use up to 34 GB of GPU memory, and the number of units run in parallel per card is configurable. `robotensor doctor` checks a host before it goes live, and an image build (`docker/build.sh`) covers rented GPU pods. Code: [robotensor-subnet](#), [vector-orchestrator](#), [RoboTwin-Vector](#).

Appendix B. The contract

Everything a third party needs to re-derive a duel’s units and check its record is a pure function of the contract below, the record and one chain block.

B.1 The contract, spec version 1

```
{
  "version": 1,
  "benchmark": { "repo": "robotensor/RoboTwin-Vector", "branch": "vector" },
  "level": 1,
  "tasks": [
    "beat_block_hammer", "move_stapler_pad", "place_bread_skillet", "place_burger_fries",
    "place_can_basket", "place_cans_plasticbox", "place_container_plate", "place_dual_shoes",
    "place_empty_cup", "place_fan", "place_mouse_pad", "place_object_basket",
    "place_object_scale", "place_object_stand", "place_phone_stand", "place_shoe"
  ],
  "runs_per_task": 10,
  "step_ratio": 2,
  "crown": { "margin": 3.0, "max_void": 0.2 },
  "model": {
    "architecture": "vector_v1.1",
    "tensors_sha256": "d652e1c440ffa90e0a06320e6d72bf0d5932e9b4688ba26fa7eefef2161ebea4"
  },
  "budgets": {
    "policy_start": 600, "act": 60, "materialize": 2400, "unit": 1800,
    "policy": 1200, "duel": 86400, "stall": 45, "stall_retries": 2
  }
}
```

Budgets are in seconds, except stall_retries, which is a count. `tensors_sha256` is the digest of the architecture's tensor manifest: every name, shape and dtype a submission must hold.

B.2 The commitment

```
commitment = "vector:" repo "@" commit "." digest
repo       = part "/" part                ; part = [A-Za-z0-9][A-Za-z0-9._-]*, repo ≤ 49 characters
commit     = 27 base64url characters      ; the revision's 20 bytes, unpadded
digest     = 43 base64url characters      ; sha256 of model.safetensors, unpadded
```

The whole string is at most 128 bytes, the chain's limit for a raw commitment. The revision is always a full commit, never a branch, so what was committed cannot change after its block.

B.3 Identifiers and units

All hashes are sha256 of UTF-8 text, written as lowercase hex, with fields joined by |.

Dealing units. For each task in contract order, a counter yields integers from $\text{sha256}(\text{vector}/\text{units}/3 \mid \text{duel id} \mid \text{entropy} \mid \text{task} \mid n)$, $n = 0, 1, 2, \dots$, taking the first 8 bytes big-endian and rejecting draws that would bias the modulo. Each run of the task takes 20 candidate scoring scenes, then 20 candidate demonstration scenes distinct from them, all in $[2,000,000, 2^{31} - 1)$. The expert's first success in each list is used, and a unit is void only if every candidate in a list fails.

Table B1. Identifiers and their derivation.

Identifier	Derivation
Submission key	First 16 hex characters of sha256(repo@revision), the revision as 40 lowercase hex characters
Duel id	sha256(spec version, challenger key, challenger revision, king key or none , king revision or none)
Event id	sha256("event" kind sequence duel id), where sequence is the validator's own event counter; for a vacate, the last field is the king's submission key
Entropy	<seed block>:0x<seed block hash>, the hash as 64 lowercase hex digits

Checking a record. Read the seed block's hash from the chain, recompute the duel id from the record's two submissions and the contract version, deal the units, and compare each unit's published scoring-scene seed with its dealt candidates. Because a task's units are drawn in order, a smaller duel's units for each task are the first units of that task in a larger one.

References

- Adaptive Agent Team (2023). Human-Timescale Adaptation in an Open-Ended Task Space. ICML 2023.
- Bittensor documentation. *Emissions*. bittensor.com/docs/concepts/emissions
- Black, K. et al. (2025a). π_0 : A Vision-Language-Action Flow Model for General Robot Control. RSS 2025.
- Black, K. et al. (2025b). $\pi_{0.5}$: a Vision-Language-Action Model with Open-World Generalization. CoRL 2025.
- Brohan, A. et al. (2023a). RT-1: Robotics Transformer for Real-World Control at Scale. RSS 2023.
- Brohan, A. et al. (2023b). RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control. CoRL 2023.
- Brown, T. B. et al. (2020). Language Models are Few-Shot Learners. NeurIPS 2020.
- Chen, T. et al. (2026). RoboTwin 2.0: A Scalable Data Generator and Benchmark with Strong Domain Randomization for Robust Bimanual Robotic Manipulation. ICML 2026.
- Chi, C. et al. (2023). Diffusion Policy: Visuomotor Policy Learning via Action Diffusion. RSS 2023.
- Chiang, W.-L. et al. (2024). Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference. ICML 2024.
- Dasari, S. and Gupta, A. (2020). Transformers for One-Shot Visual Imitation. CoRL 2020.
- Di Palo, N. and Johns, E. (2024). Keypoint Action Tokens Enable In-Context Imitation Learning in Robotics. RSS 2024.
- Dosovitskiy, A. et al. (2021). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. ICLR 2021.
- Du, Y. et al. (2023). Learning Universal Policies via Text-Guided Video Generation. NeurIPS 2023.
- Duan, Y. et al. (2016). RL^2 : Fast Reinforcement Learning via Slow Reinforcement Learning. arXiv preprint.
- Duan, Y. et al. (2017). One-Shot Imitation Learning. NIPS 2017.
- Finn, C., Abbeel, P. and Levine, S. (2017a). Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. ICML 2017.
- Finn, C. et al. (2017b). One-Shot Visual Imitation Learning via Meta-Learning. CoRL 2017.
- Fu, L. et al. (2025). ICRT: In-Context Imitation Learning via Next-Token Prediction. ICRA 2025.
- Generalist AI (2025). GEN-0: Embodied Foundation Models That Scale with Physical Interaction. Company report, 4 November 2025.
- Generalist AI (2026). GEN-1.5: Embodied Foundation Models are One-Shot Learners. Company report, 19 August 2026.
- Ho, J., Jain, A. and Abbeel, P. (2020). Denoising Diffusion Probabilistic Models. NeurIPS 2020.
- Jain, V. et al. (2024). Vid2Robot: End-to-end Video-conditioned Policy Learning with Cross-Attention Transformers. RSS 2024.
- James, S. et al. (2020). RLBench: The Robot Learning Benchmark & Learning Environment. *IEEE Robotics and Automation Letters* 5.
- Jang, E. et al. (2021). BC-Z: Zero-Shot Task Generalization with Robotic Imitation Learning. CoRL 2021.
- Janner, M. et al. (2022). Planning with Diffusion for Flexible Behavior Synthesis. ICML 2022.
- Kiela, D. et al. (2021). Dynabench: Rethinking Benchmarking in NLP. NAACL 2021.
- Kim, M. J. et al. (2024). OpenVLA: An Open-Source Vision-Language-Action Model. CoRL 2024.
- Laskin, M. et al. (2023). In-context Reinforcement Learning with Algorithm Distillation. ICLR 2023.
- Lee, J. et al. (2019). Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks. ICML 2019.
- Li, C. et al. (2024). BEHAVIOR-1K: A Human-Centered, Embodied AI Benchmark with 1,000 Everyday Activities and Realistic Simulation. arXiv preprint; preliminary version at CoRL 2022.
- Lipman, Y. et al. (2023). Flow Matching for Generative Modeling. ICLR 2023.

- Liu, B. et al. (2023). [LIBERO: Benchmarking Knowledge Transfer for Lifelong Robot Learning](#). NeurIPS 2023 Datasets and Benchmarks.
- Mees, O. et al. (2022). [CALVIN: A Benchmark for Language-Conditioned Policy Learning for Long-Horizon Robot Manipulation Tasks](#). *IEEE Robotics and Automation Letters* 7.
- Nasiriany, S. et al. (2024). [RoboCasa: Large-Scale Simulation of Everyday Tasks for Generalist Robots](#). RSS 2024.
- Octo Model Team (2024). [Octo: An Open-Source Generalist Robot Policy](#). RSS 2024.
- Open X-Embodiment Collaboration (2024). [Open X-Embodiment: Robotic Learning Datasets and RT-X Models](#). ICRA 2024.
- Pan, J. et al. (2023). [What In-Context Learning “Learns” In-Context: Disentangling Task Recognition and Task Learning](#). Findings of ACL 2023.
- Patel, A., Pekarek, B., Castro Hernandez, J. E. and Song, S. (2026). [Behavior Prompting Policy: Demonstrations as Prompts for Manipulation](#). arXiv:2606.30457.
- Perez, E. et al. (2018). [FiLM: Visual Reasoning with a General Conditioning Layer](#). AAAI 2018.
- Radford, A. et al. (2021). [Learning Transferable Visual Models From Natural Language Supervision](#). ICML 2021.
- Rao, Y. (n.d.). [Bittensor: A Peer-to-Peer Intelligence Market](#). Whitepaper.
- Ronneberger, O., Fischer, P. and Brox, T. (2015). [U-Net: Convolutional Networks for Biomedical Image Segmentation](#). MICCAI 2015.
- Skild AI (2025). [The case for an omni-bodied robot brain](#). Company report, 24 September 2025.
- Skild AI (2026). [Introducing S1: In-Context Learning for Robotics](#). Company report, August 2026.
- Song, J., Meng, C. and Ermon, S. (2021). [Denoising Diffusion Implicit Models](#). ICLR 2021.
- Sun, Y. et al. (2020). [Test-Time Training with Self-Supervision for Generalization under Distribution Shifts](#). ICML 2020.
- Teutonic (2026). [A king-of-the-hill pretraining system for Bittensor subnet 3](#). Source code.
- Wang, C. et al. (2023). [MimicPlay: Long-Horizon Imitation Learning by Watching Human Play](#). CoRL 2023.
- Wang, L. et al. (2024). [GenSim: Generating Robotic Simulation Tasks via Large Language Models](#). ICLR 2024.
- Wu, H. et al. (2024). [Unleashing Large-Scale Video Generative Pre-training for Visual Robot Manipulation](#). ICLR 2024.
- Xiao, G. et al. (2024). [Efficient Streaming Language Models with Attention Sinks](#). ICLR 2024.
- Xiong, R. et al. (2020). [On Layer Normalization in the Transformer Architecture](#). ICML 2020.
- Ye, S. et al. (2026). [World Action Models are Zero-shot Policies](#). arXiv preprint.
- Yin, Y. et al. (2025). [In-Context Learning Enables Robot Action Prediction in LLMs](#). ICRA 2025.
- Yu, T. et al. (2019). [Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning](#). CoRL 2019.
- Zhao, T. Z. et al. (2023). [Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware](#). RSS 2023.
- Zhou, J. et al. (2026). [Zero-WAM: In-Context World-Action Modeling from Human Videos for Open-Ended Task Generalization](#). arXiv preprint.
- Zhou, Y. et al. (2019). [On the Continuity of Rotation Representations in Neural Networks](#). CVPR 2019.