Learned skill policies on top of classical perception for the TOWER Challenge

Team Sakai — technical report for result verification — November 2026

Summary. Our final policy, **RAX-AM** ("RAX Action Model"), solves all five TOWER tasks with one design: classical RGB perception estimates the pose offsets of the next block and its destination from the two static cameras; a **learned skill selector** decides which skill is active and when it ends; a **learned executor** outputs the next 16 joint-command rows; a classical safety filter can only hold or slow the commands. The networks are small MLPs (0.45–17 M parameters) trained by behaviour cloning on the official training demonstrations, augmented by kinematic retargeting of those demonstrations. Earlier in the challenge we also trained a π0.5 vision-language-action model on Tasks 1–3 (MT3a2) and a hand-designed program (RAX program), which served as the source of the perception and retargeting code. Per-variant best over our official submissions: **93.58**; using only learned submissions (RAX-AM and π0.5): **93.50**. All code, weights and launch commands are provided, and a script recomputes the official model identity of every submission from them.

1. Submissions covered by this report

Submission	Model name	Policy	Variants in our per-variant best	Successes	Score
`66f42b37`	all10-rax-am-v6d-hk	RAX-AM	T1-2, T1-3, T2-3, T3-2, T3-3, T4-3, T5-2	91/100	91.93
`56e5dc08`	all10-rax-am-v6b-hk	RAX-AM	T4-2	89/100	90.65
`069db045`	task123-pi05-MT3a2-...-interp30-v1-hk	π0.5 VLA (Tasks 1–3)	T2-2	48/60	—
`fcea9688`	all10-rax-program-v1-hk	RAX program	T5-3	90/100	90.59

Table 1. Submissions that contribute to our per-variant best (track `tower-sim-all`, 30 Hz, 10 episodes per variant). Score = single-submission overall score with the official formula. v6b and v6d differ only in the Task-4 models.

2. Interface and serving (shared by all submissions)

Each policy is a websocket server implementing `tower-infer-v1`. Per request it reads the official observation (four RGB images, 16 joint positions, 2 gripper widths, task variant, step index) and returns 16 rows of 18-D absolute commands at 30 Hz. A thin dispatcher routes each episode to the per-task backend by the `X-Tower-Task-Variant` header and forwards messages unchanged. Every backend freezes a manifest of the sha256 of all its files and settings and refuses to start if anything changes; the dispatcher's identity is the hash of its per-variant backend table, which is the `model_revision` recorded by the official site. RAX-AM and the RAX program run on CPU (numpy only, 20–40 ms per request); MT3a2 runs on one GPU.

3. RAX-AM (main method)

3.1 Architecture

RAX-AM decomposes each task into a sequence of skills taken from the phase labels of the demonstrations (Tasks 1–3: approach, close, lift, transport, place, release, retreat, settle, park; Task 4: approach, push, retreat, settle per block; Task 5: the phase events of each target). One request is processed as follows:

- **Perception (classical).** Once per block operation (at the start of its approach skill; the destination at episode start, and in Task 5 again before the second insertion), a gradient-edge template fit on the left-back and right-front RGB images estimates the planar offset (dx, dy, dyaw) of the target block relative to a reference pose taken from one training demonstration (search range ±22 mm / ±12°), and the translation of the destination. Wrist cameras are not used, except for a blank-image check.
- **Skill selector (learned).** An MLP maps low-dimensional features — joint positions, gripper widths, counters of the policy's own past gripper commands, rows spent in the current skill and

the perception offsets — to the current skill, the arm for the next operation (Tasks 1–3), and the end of the skill (Task 4: boundary class; Task 5: skill class plus a regression of the remaining rows).

- **Executor (learned).** An MLP conditioned on the skill, the rows spent in it and the perception offsets outputs the next 16 command rows as increments. Tasks 1–3: increments to the policy's own last command (8 columns of the working arm); this executor does not read the measured joints, which enter only through the selector. Tasks 4–5: increments to the measured joint positions (all 16 columns), with the measured state as input.
- **Safety filter (classical).** Joint limits, a per-row step limit and finite-value checks. A rejected chunk is replaced by holding the current command, then by a rate-limited version of the network output; it never substitutes stored trajectories.

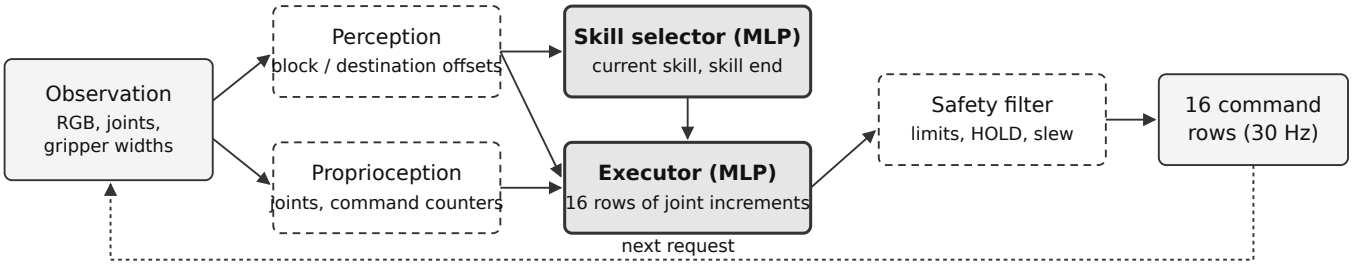


Figure 1. RAX-AM inference for one request. Bold boxes are learned; dashed boxes are classical.

Family	Selector	Executor	Training rows	Training
Tasks 1–3 (one model for 6 variants)	MLP 104-512-512-256-13, 0.45 M	MLP 53-1024-1024-1024-512-128, 2.75 M	2.1 M (exec), 3.3 M (sel)	30 / 12 epochs, seed 0
Task 4, per variant	MLP 47-512×3-(17+6), 0.56 M	MLP 47-1024×4-256, SiLU, 3.46 M	0.93–0.97 M	40k steps, batch 4096, input noise 0.012
Task 5, per variant	cls + remaining-rows MLPs, 3.2 M each	MLP, width 1024, 4 blocks, 17.2 M	44 demos × 21	40k steps, batch 4096, lr 1e-3

Table 2. RAX-AM networks. Weights were trained with PyTorch and exported to numpy; inference uses numpy only.

3.2 Training data

All models were trained on the official TOWER-SimData demonstrations, **training split only** (Tasks 1–3: 40–41 episodes per variant; Tasks 4–5: 44 per variant, 6 held out for validation). No data from running a policy in simulation was used. Two kinds of augmentation were added:

- **Kinematic retargeting.** For each demonstration, the tool path of each skill segment is rigidly moved to a random source offset (±20–22 mm, ±11–12°) and destination offset (±5–8 mm), and joint rows are re-solved by inverse kinematics on the arm's kinematic chain (8 copies per episode for Tasks 1–3, 20 for Task 5, comparable for Task 4). This code is shared with the RAX program (Section 5). It teaches the executor to follow perceived offsets rarely seen in the demonstrations.
- **Task 4, v6d only (model v7p-m).** Demonstrations were edited by `pulse_teacher.py` with rule set `rules_m.json` (the working arm lifts slightly before the final push and pushes in segments) and then retargeted (1364 / 2024 trajectories for 4-2 / 4-3). The trainer and the network are the same as for v6b.

Labels: skills and boundaries come from the demonstration phase annotations (`sim/task_phase`, event lists); block poses from the simulator state are used only to build training targets. At inference the policy reads only the official observation.

3.3 Task-specific classical components

The table lists every non-learned part that can change the commands, as requested by the organisers.

Component	Where	What it does	Type
Perception	all tasks	Template fit of block / destination offsets on static RGB images; input to the networks	classical vision
Kinematics	all tasks	Forward kinematics and Jacobians of the arm chains for features and checks. Inverse kinematics is used only to build training data, not at inference	classical

Safety filter / HOLD	all tasks	Limits, step check; on rejection repeats the last command	supporting controller
Rate limiting	T4, T5	After repeated HOLDs, only the first row is executed and slewed (T4); slew of rejected chunks (T5); a cap on the gripper closing speed taken from the demonstrated closing profile (T5)	action buffering
Start transition	T1-2	If the start pose differs from the demonstrated start by >0.03 rad, a 45-row cosine interpolation to the median first command of the training demonstrations (one stored row), then the learned policy	scripted routine (45 rows)
Height loop	T5-3; T5-2 only when the source offset >6 mm / 2°	Proportional correction of the arm height during alignment and insertion, from forward kinematics, towards the per-skill median of the demonstrations	supporting controller
Slow window, slot measurement, drop compensation	T5-3 (v6b/v6d bundle v2s.9c-h; Section 3.4)	Fixed 3× slow-down of one push window; lateral correction ≤4 mm from an RGB corner measurement of the slot; open-loop height profile ≤3 mm	scripted / classical
Fixed schedule	T4 bundles	Mean skill durations; used only in the `fixed` selector ablation. The evaluated servers ran `--selector-mode learned`	not used
Trajectory reuse	—	None at inference: apart from the T1-2 start row and joint-preload constants, bundles store no demonstrated command rows	—

Table 3. Non-learned components of RAX-AM. File and function names are listed in the repository README and MAPPING.

Evidence that the learned parts matter. In our local Isaac Sim replica (built from the simulator configuration in the official HDF5 files, used only for development): zeroing the perception input dropped T1-2 + T3-2 from 9/10 to 2/10 and raised the Task-4 push error from 0.4 to 12 mm; replacing the learned selector by fixed durations lowered T4-2 from 2/3 to 1/3 (weak evidence, the demonstrated durations are very regular).

3.4 The four classical modules of Task 5-3

Task 5-3 (bundle `rax-am-t5-v2s.9c-h`, used by v6b and v6d) adds four non-learned modules to the learned selector and executor. Two of them (height loop, drop compensation) add a vertical tool offset c to the working arm's rows as $dq = c \cdot d$, where d is the damped least-squares joint direction that moves the tool point straight up with its orientation held; for these two the learned models are shown the measured joints with $c \cdot d$ subtracted, so that the executor does not compensate the offset a second time. All four keep the unchanged safety filter.

Module (file)	Active	Input	What it does	Bounds and constants
Height loop (`am_height.py`)	Both targets, while the block is held in the *align* and *insert* skills	Measured joints (forward kinematics)	Estimates the height of the block's leading end from the tool height relative to the grasp and the gripper pitch; integral correction towards a per-skill target height, once per 16-row chunk; faded to zero over 32 rows from *release*	Gain 0.5 per chunk, step ≤0.75 mm, |c| ≤ 4 mm. Target heights = median over the 44 training demonstrations
Slow window (`am_slow.py`)	First target, *push* skill, skill progress 128–192 rows	None (skill progress only)	Executes the executor's 16 rows at one third speed (each chunk covers 16/3 model rows, interpolated); the models' clocks keep demonstration time. About 2.8 s longer per episode	Speed 1/3; fixed window
Drop compensation (`am_drop.py`)	First target, same window	None	Open-loop vertical profile: 0 at progress 128, linearly to −3 mm at 160, held to 192, then back to 0 over 32 rows (the slowed arm lags less and rides higher)	−3 mm, fixed profile

Slot measurement (`am_slot.py`)	Once, at the start of the second target	Left-back RGB image	Locates the two near corners of the slot by the brightness step at their vertical edges; if the predicted side clearance of the planned path is below 0.35 mm, shifts the lateral destination offset given to the executor to leave 1 mm per side	Shift ≤4 mm; no change if a corner is not found clearly or the slot is narrower than the block

Table 3a. Task 5-3 classical modules. The window and profile values come from our replica; `am_slow.py` also lists a Task 5-2 window, which is inactive because Task 5-2 is served by another bundle.

Origin and evidence. The two failure modes they target were identified in the official result videos of our earlier submissions and reproduced in the replica: during the first push, while the block tips flat and its nose crosses onto the far side block, it drags the roof or a side wall along (slow window, drop compensation); the second block hits a near corner of a slot whose side wall was moved by the first push (slot measurement). The height loop targets a held block that meets the wall during insertion because its height is off. In the replica, the height loop reduced those collisions from 10 episodes to 0, and the slow window removed the failures of the crossing type (4/55 without it, 0/26 and 0/23 for the two versions with it); the final choice between the versions with and without drop compensation was not statistically significant (Fisher p = 0.38). On the frames of 20 earlier official Task 5-3 episodes, the slot measurement never triggered (0 of 140 frames), so it probably had no effect in the evaluation. Officially, Task 5-3 went from 5/10 (score 66.4, v3: height loop only) to 8/10 (85.1, v6b/v6d) with the same learned weights; the versions also differ in the safety-filter dead-lock fix, so the gain cannot be assigned to a single module.

4. π0.5 multitask VLA for Tasks 1–3 (MT3a2)

Model. π0.5 (openpi, commit `215abfb`, unmodified) full fine-tune, 3.35 B parameters, action horizon 16, action dim 32. Inputs: left-back, left-wrist and back-wrist images (224 px), prompt = official instruction + “(two-layer variant)” / “(three-layer variant)”, state = 16 joints + 2 lifter values (q01/q99-normalised) + 3 counters of the policy's own gripper events, zero-padded to 32.

Training. 241 training episodes of the six Task 1–3 variants (official split; 42 val / 48 test unused). Stage 1: 38.4 k steps from `pi05_base`, lr 2e-5; stage 2: 19.2 k steps, lr 1e-5; run A: 16.8 k steps adding the other two phases of the 30 Hz data (frames 1::3 and 2::3); run A+ (submitted): 16.8 k steps from A, peak lr 5e-6 cosine to 5e-7, AdamW, batch 32 on 4×H20, lighting augmentation p = 0.5, proportional task sampling.

Inference. *`mean8`*: 8 flow samples (10 steps each) with noise derived from the episode seed, averaged after de-normalisation. *`camfix`*: the two static images of the official Task 1–3 track are rotated by 180° to match the training data orientation. *`interp30`*: the model is queried every 48 steps; its 16 nodes (10 Hz) are linearly interpolated to 48 rows at 30 Hz, 16 rows per reply. No hold, grip guard or scripted routine.

5. RAX program (no learned weights)

The RAX program was our earlier hand-designed policy and the origin of RAX-AM's perception and retargeting code. For each variant it uses one training demonstration as a template, cut into skill segments. For each segment it perceives the offsets as in Section 3.1, rigidly moves the template tool path accordingly and re-solves every row with inverse kinematics; it verifies grasps (gripper width) or pushes (image correlation) and may retry once; Task 5 adds a forward- kinematics position correction. This is **perception-conditioned trajectory reuse**: timing, speeds and the idle arm follow the template, and in a scene with zero offset the output equals the template commands (Task 4 also executes the first 93 template rows unchanged). It is closed-loop with respect to block poses but contains no learned component. It contributes only T5-3 (85.65 vs 85.15 for RAX-AM) and is counted in our per-variant best of 93.58. Because it has no learned component, we also propose an alternative result without it: **93.50**, with T5-3 taken from v6d (85.15), so that every variant comes from a learned submission (Appendix A). We leave the choice between the two to the organisers.

6. Results by task

Variant	v6d	v6b	MT3a2	RAX prog.	Best	Best, learned
T1-2 / T1-3	100 / 100	100 / 100	85.1 / 96.1	100 / 100	100 / 100	100 / 100
T2-2 / T2-3	91.7 / 100	91.7 / 100	100 / 91.1	91.4 / 100	100 / 100	100 / 100
T3-2 / T3-3	100 / 100	100 / 95.6	88.3 / 75.6	100 / 95.6	100 / 100	100 / 100
T4-2 / T4-3	77.3 / 91.9	83.9 / 79.2	—	80.9 / 81.3	83.9 / 91.9	83.9 / 91.9
T5-2 / T5-3	100 / 85.1	100 / 85.1	—	100 / 85.7	100 / 85.7	100 / 85.1
Overall	**91.93**	**90.65**	**—**	**90.59**	**93.58**	**93.50**

Table 4. Official variant scores s = 60·LHPR + 30·TSR + 10·P/(1+ODR); overall = weighted mean with weight 3 for Tasks 4–5. Successes, sub-metrics and the effect of each selection: Appendix A.

Tasks 1–3. RAX-AM succeeds in 59 (v6d) and 58 (v6b) of 60 episodes. From the official videos, MT3a2's failures are mainly hesitation around the gripper-closing decision and placement errors when the source block is displaced. **Task 4.** From the official videos of earlier submissions, most failures come from a late push dragging a block of a lower layer (L1_2 in 4-2, L2_2 in 4-3) so that it ends on the base. The v7p-m executor (v6d) scored 9/10 on 4-3 but 5/10 on 4-2, against 6/10 and 7/10 for v6b. **Task 5.** 5-2 succeeded in 10/10 episodes and 5-3 in 8/10 for both RAX-AM versions.

7. Known limitations

- Perception is a hand-designed template fit tuned to the simulator's rendering; it is the component most likely to fail under appearance changes, and its search range (±22 mm, ±12°) bounds the offsets the policy can handle.
- The executors imitate demonstrations and their retargeted copies; they do not recover well from states far from the demonstrated ones (e.g. a dragged lower block in Task 4).
- Several Task 5-3 corrections (Table 3) are classical and tuned on our replica; the T1-2 start transition is scripted.
- Our replica was calibrated against official results only for Tasks 1–3; for Tasks 4–5 its ranking of candidates was unreliable (it predicted v7p-m better on 4-2 and worse on 4-3; the official results show the opposite).
- Ten episodes per variant leave large uncertainty: one episode changes a variant score by up to about 10 points.

8. Data use and development process

Training data. Every submission was trained only on the official TOWER-SimData demonstrations, training split:

Submission	Demonstrations used	Augmentation / processing
RAX-AM, Tasks 1–3	40–41 per variant, one model for the six variants	Kinematic retargeting, 8 copies per episode
RAX-AM, Task 4	44 per variant (6 more held out for validation)	Retargeting; v6d only: demonstrations edited by `pulse_teacher.py` / `rules_m.json` before retargeting (1364 / 2024 trajectories)
RAX-AM, Task 5	44 per variant (6 more held out for validation)	Retargeting, 20 copies per episode
MT3a2 (π0.5)	241 episodes of the six Task 1–3 variants	The other two 30 Hz phases (frames 1::3, 2::3); lighting augmentation p = 0.5; openpi image augmentation
RAX program	One template demonstration per variant (e.g. T5-3: `episode_000003`)	None offline; the template is retargeted online (Section 5)

Table 5. Training data. Skill labels come from the demonstrations' phase annotations; simulator block poses in the HDF5 files are used only to build training targets.

Not used for training by any submission: the validation and test splits (validation episodes only for model selection), data from running our own policies in simulation, and any official evaluation episode, observation, video or score.

Development only. (1) A local Isaac Sim replica of the evaluation simulator built from the configuration stored in the official HDF5 files (joint gains, latency, gripper model, success tolerances) and the official stage; it was calibrated against our official Task 1–3 results, the Task 4–5 replicas

were not. Candidates were compared on paired replica scenes before each submission, and the classical Task 5-3 components were tuned there. (2) Camera intrinsics from SimData; the principal points of the two static cameras were refined on training-split renders. (3) The observations our servers received during official evaluation, and later the official result videos (as permitted by the organisers), were used to diagnose failures. Fixes that came from them: the Task 1–3 static-camera orientation for MT3a2, the T1-2 start transition for RAX-AM, and the failure modes addressed by the Task 5-3 modules (Section 3.4), whose slot detector was also checked on official frames. (4) Official scores were used to choose between our versions, as the per-variant best rule does (e.g. T4-2 from v6b, T4-3 from v6d); our final selection is therefore not blind to the official results.

9. Reproducibility

The repository contains the evaluated files at their original paths, the systemd unit definitions and package lists, and launch scripts; weights are on Hugging Face with sha256 for every file. `tools/verify_package.py` recomputes the identity of each backend and dispatcher and checks every file hash; it reproduces all four official `model_revision` values. We also started every RAX-AM and RAX-program server from the package alone in an isolated namespace and passed a qualification-style probe on all ten variants. The evaluated processes ran unchanged from before their evaluation until packaging. Differences (launch wrappers, removed helper scripts, omitted logs and preprocessed data) are listed in `DIFFERENCES.md`; none affects a hashed file.

Appendix A. Official results per variant and the per-variant selection

All numbers are recomputed with the official formula from the official result files of the four submissions (track `tower-sim-all`, protocol `tower-paper-v1`, 10 episodes per variant). Cells give the variant score and, in brackets, the number of successful episodes. **Bold** marks the entry counted in our per-variant best. "Δ overall" is the change of the overall score against using v6d alone (weight 1 for Tasks 1–3, 3 for Tasks 4–5, divided by 18).

Variant	w	v6d	v6b	MT3a2	RAX prog.	Counted	Δ overall
T1-2	1	**100 (10)**	100 (10)	85.08 (8)	100 (10)	v6d (tie)	0
T1-3	1	**100 (10)**	100 (10)	96.11 (9)	100 (10)	v6d (tie)	0
T2-2	1	91.67 (9)	91.67 (9)	**100 (10)**	91.43 (9)	MT3a2	+0.46
T2-3	1	**100 (10)**	100 (10)	91.11 (8)	100 (10)	v6d (tie)	0
T3-2	1	**100 (10)**	100 (10)	88.33 (8)	100 (10)	v6d (tie)	0
T3-3	1	**100 (10)**	95.56 (9)	75.56 (5)	95.56 (9)	v6d	0
T4-2	3	77.29 (5)	**83.91 (7)**	—	80.87 (7)	v6b	+1.10
T4-3	3	**91.94 (9)**	79.17 (6)	—	81.33 (7)	v6d	0
T5-2	3	**100 (10)**	100 (10)	—	100 (10)	v6d (tie)	0
T5-3	3	85.15 (8)	85.06 (8)	—	**85.65 (8)**	RAX prog.	+0.08
Overall		**91.93**	**90.65**	**—**	**90.59**	**best 93.58; learned 93.50**	**+1.65**

Table A1. Variant scores of the four submissions. Evaluated: MT3a2 and RAX program on 30 Oct 2026, v6b and v6d on 3 Nov 2026. MT3a2 ran Tasks 1–3 only. Ties at 100 (v6d, v6b, RAX program) are attributed to v6d; the best learned entry for T5-3 is v6d.

Variant	Counted submission	Successes	LHPR	TSR	ODR	Score
T1-2, T1-3, T2-3, T3-2, T3-3, T5-2	v6d	10/10 each	1.000	1.0	0.000	100.00
T2-2	MT3a2	10/10	1.000	1.0	0.000	100.00
T4-2	v6b	7/10	0.933	0.7	0.237	83.91
T4-3	v6d	9/10	0.978	0.9	0.517	91.94
T5-3	RAX program	8/10	0.925	0.8	0.436	85.65
T5-3 (learned alternative)	v6d	8/10	0.925	0.8	0.564	85.15

Table A2. Official metrics of the counted entries. Score s = 60·LHPR + 30·TSR + 10·P/(1+ODR), P = (2·LHPR + TSR)/3. Result hashes (`result_sha256`): v6d `087b1b2b…`, v6b `53b0bfc3…`, MT3a2 `6af98289…`, RAX program `2e2635f9…`.

What the selection does. Three choices change the overall score against v6d alone. (1) T4-2 from v6b (+1.10): the two versions differ only in the Task 4 model (Section 3.2), and the edited-demonstration model of v6d was better on 4-3 but worse on 4-2. (2) T2-2 from MT3a2 (+0.46): 10/10 against 9/10 for RAX-AM. (3) T5-3 from the RAX program (+0.08): the same successes and progress as RAX-AM; only the disturbance rate is lower (0.436 against 0.564). Our proposed alternative without the RAX program is 93.50, with every variant covered by a learned submission (T5-3 from v6d).

Evaluation noise. v6b and v6d run the same Task 1–3 and Task 5 backends with identical files, yet T3-3 gave 10/10 and 9/10 and T5-3 85.15 and 85.06. With 10 episodes per variant, differences of one episode, including the T2-2 and T5-3 selections above, are within the run-to-run variation of a single policy.