When Auto-Mined Robot Skills Help: A CaP-X-Style Ablation of Executable Skill Libraries

realkim93
auto-capx (independent), Seoul, Republic of Korea
CaP-X Team
Correspondence: pheonix37@naver.com

AI assistance: Anthropic Claude (Opus 4.7, 1M context) was used as a coding and writing assistant; see Acknowledgements.

***Abstract*—We test whether an LLM-controlled robot code-generation agent benefits from executable skills mined from its own prior programs. NVIDIA CaP-X already includes a manually-curated helper library (researchers selected $\sim$9 helpers from observed LLM trial patterns and added them to the published S3 reduced-API configuration); the question we ask is whether that human curation step can be replaced by an automatic mining loop. Our *auto-capx* extension to a Code-as-Policies-style `cube_lifting` pipeline does so: generated Python functions are extracted by AST, filtered by quality gates, optionally deduplicated, and reinjected into later prompts and execution namespaces – no human in the loop. The north-star hypothesis is that an algorithmically-curated mined-skill library should beat a no-skills baseline (P21_a, ablating all helpers).**

The strongest result is conditional. With `gpt-4.1`, the no-skill baseline (P21_A) reaches $39/50 = 78.0\%$ (Wilson 95% CI $[64.8\%, 87.2\%]$). Two no-dedup mined-skill libraries reach $67/70 = 95.7\%$ for the namespace-enabled 16-skill arm (C1, CI $[88.1\%, 98.5\%]$) and $68/70 = 97.1\%$ for the gated 14-skill arm (C2, CI $[90.2\%, 99.2\%]$), separating from the baseline by roughly $+18$–19pp. A typed empty-stub control (EMPTY_NS) reaches only $13/30 = 43.3\%$, showing that executable function bodies, not merely a typed namespace, drive the effect.

The production-style library does not preserve this gain. The structural-hash deduplicated library (C3v2) reaches $58/70 = 82.9\%$, statistically indistinguishable from the no-skill baseline and -14pp below the no-dedup gated library. A quality-ranked same-size 11-skill library (MANUAL_11), formed by dropping the three lowest-quality C2 skills, reaches $66/70 = 94.3\%$, suggesting that the loss is caused by survivor selection rather than by the number of skills. Docstring controlled tests further show that LLMs strongly prefer documented variants when task-fit-equivalent functions compete.

Thus v1 supports a bounded claim: auto-mined executable skills improve this CaP-X-derived `cube_lifting` pipeline under no-dedup selection, but the current dedup survivor rule can erase the benefit.

A v2 follow-up resolves the two largest open boundaries with two real-run additions (n=50 each on Claude/DeepSeek baselines + library, n=70 each on three Dedup v3 cutoffs; we report the auto-capx-defined "Task completed" rate – the trial's final attempt being a success – so v1 and v2 numbers are directly comparable). *Multi-backbone library effect, fully closed:* on Claude Sonnet 4 the library effect is $+12$pp (94.0% [89.5, 99.6] vs 86.0% [73.8, 93.0]; CIs overlap by 3.2pp at the boundary, but the one-sided binomial test $P[X \geq 49 \mid n{=}50, p{=}0.86] \approx 0.5\%$); on DeepSeek v3 it is $+88.0$pp (94.0% [83.8, 97.9] vs 6.0% [2.1, 16.2]; CIs fully separated, $P \approx 10^{-54}$). The DeepSeek finding is the largest single-task library benefit observed in this project and demonstrates that library magnitude is baseline-dependent (the weakest baseline benefits most). *Dedup v3 algorithmic robustness:* top-k-by-quality_score at n=70 each reaches $91.4\%/87.1\%/97.1\%$ for $k \in \{10, 12, 13\}$. Combined with k=11 (94.3% from paper v1's manual-11 arm), k=11 and k=13 are statistically separated from the production structural-dedup library C3v2 (82.9%, one-sided $P{=}10^{-3}$), k=10 is marginally above ($P{=}0.034$), and k=12 is not separated ($P{=}0.22$). The smart dedup is therefore an algorithmic recipe with non-monotone k-sensitivity, not a uniformly robust function. Three additional "smoke" micro-evaluations (Section IV-L) probe the boundaries of these claims: (i) a new `cube_stack_3` task floors at $0/15$ for both no-skills and library arms, with the bottleneck localised to a vision-pipeline mask-fragmentation regime that is task-specific rather than library-related, and the library still confers a 24–34% *code-efficiency* reduction even at the floor; (ii) a round-2 mining attempt on a mid-range library returns 0 promotable new skills, evidencing a "namespace saturation" where dense libraries induce purely imperative LLM code with no further function-level abstractions to extract; (iii) a verbose mechanism trace (n=3 each on Claude / DeepSeek) suggests the multi-backbone library asymmetry tracks self-correction ability rather than reasoning ability – both backbones make the same first-attempt errors but diverge in regeneration convergence.

***Index Terms*—LLM agents, code-as-policies, skill libraries, ablation study, robot manipulation, CaP-X.**

I. Introduction

NVIDIA CaP-X [1] studies coding agents for robot manipulation: an LLM receives a small robot API plus a manually-curated set of helper skills (the published S3 configuration uses $\sim$9 helpers selected by the authors from observed LLM trial code), emits executable Python, runs the code in simulation, and receives visual feedback. This paper keeps that observable code-generation loop but asks a narrower question: can the manual helper-curation step itself be replaced by an automatic mining loop that turns the agent's own past robot-control programs into a reusable executable skill library?

The proposed extension, *auto-capx*, mines named Python functions from previous generated code, filters them by execu-

tion and code-quality signals, and reinjects promoted functions into both the prompt and the execution namespace. The north-star hypothesis is simple: if the mined functions encode useful robot-control subroutines, later trials should outperform a no-skills CaP-X-style baseline. The scientific risk is equally simple: apparent gains may be caused by harness fixes, lucky samples, or survivor-selection artifacts rather than by reusable skill content.

We evaluate this hypothesis on `cube_lifting`. The claim ladder for v1 is:

1) **Measured within-task gain.** The no-dedup mined-skill library beats the no-skill baseline (P21_A) on `cube_lifting` by roughly +18–19pp with disjoint Wilson intervals.
2) **Skill bodies matter.** A typed-but-empty namespace control performs far below both baseline and real-skill conditions, so the gain is not just scaffolding.
3) **Survivor selection matters.** The structural-hash deduplicated production library falls back toward baseline, while a same-size quality-ranked manual subset (MANUAL_11) recovers most of the no-dedup performance.
4) **Transfer is not yet measured.** `cube_stack` and LIBERO floor under the present harness; Claude/DeepSeek C2 endpoints are high but lack matched no-skills baselines.

The paper's contribution is therefore not a broad claim that mined robot skills generally solve manipulation tasks. It is a bounded ablation result: executable mined skills can help a CaP-X-derived `cube_lifting` agent, but current structural dedup can erase the effect, and transfer requires a better measurement regime.

Our reported contributions are:

1) A CaP-X-derived executable skill-mining pipeline for robot code generation, including AST extraction, quality gates, deduplication, and prompt/runtime reinjection.
2) A controlled `cube_lifting` ablation showing a no-dedup skill-library win over a no-skills baseline, plus an empty-stub control isolating executable body content.
3) Evidence that the present dedup survivor rule is the main failure point: the production structural-dedup library (C3v2) loses the no-dedup gated-library (C2) gain, while the quality-ranked same-size library (MANUAL_11) recovers it at the same library size.
4) Docstring and decoy probes explaining why survivor choice affects LLM invocation.
5) Negative transfer results that delimit v1 and define the next experiment: a medium-difficulty task with neither ceiling nor floor.

II. Related Work

Code-as-policies for robots. CaP-X [1] is a recent baseline in which a small base API plus a manually-curated helper-skill library plus VDM feedback drives an LLM to produce executable robot code. Our pipeline begins from CaP-X and replaces the manual helper-curation step with an auto-mining stage; the LLM still receives a helper library each trial, but the helpers are now algorithmically extracted from the LLM's own prior code instead of researcher-selected. The original Code-as-Policies paper [2] did not include a learned library.

Agent harnesses with cumulative skills. Voyager [3] pioneered the idea of an LLM agent that grows a skill library over a long horizon in Minecraft. SWE-agent [4] and OpenHands [5] apply similar ideas to software engineering. We borrow the *plumbing* of skill mining from this lineage but examine its behaviour on a robot task with a much harder physics-grounded reward.

Skill libraries for robots. Prior work on robot skill libraries typically curates skills manually or uses RL to learn options. Our setting is closer to neural program induction: the LLM emits Python, we run AST analysis, we promote candidates that pass quality gates, and we feed the promoted set back into the next prompt.

Methodology. Our work also touches on ablation methodology for agent systems: when a high-performing run follows several simultaneous harness changes, each causal explanation must be treated as a hypothesis until isolated by controls.

III. Method

A. *auto-capx pipeline*

Figure 1 sketches the *auto-capx* pipeline. A single `cube_lifting` trial proceeds as: (i) the LLM is shown the base APIs and the current promoted skill library; (ii) the LLM emits a Python code block; (iii) the block is executed in the robosuite environment [6]; (iv) the VDM (`gemini-3.1-pro-preview`) decides whether the block succeeded, and if not, the LLM is asked to regenerate; (v) when the trial finishes, all named functions emitted across the trial are AST-walked and added to a candidate pool; (vi) candidates are scored by quality gates and structurally deduplicated, then promoted into the library that the next trial will see.

Quality gates. The gate stack has three soft gates – minimum success rate, source-task generality (a skill must occur in ≥ 2 tasks), and code quality (docstring + complexity + AST validity) – and three hard-fail gates – resolvable dependencies, non-noop body, and vision-server availability. A skill must pass an overall threshold of 0.5 on the soft side and not trigger any hard-fail to be eligible for promotion. Phase 1.4 + 1.8 (Section IV-B) measures which gates actually fire on our pool.

Dedup. The original dedup keys each candidate by a structural signature $(\text{ARG\_COUNT}, \text{HAS\_RETURN}, \mathit{frozenset}(\text{CALLED\_NAMES}))$. Within a cluster sharing a key, the original survivor rule kept the lexicographically first variant. Dedup v2 changes the survivor rule to $(\text{HAS\_DOCSTRING}, \text{SUCCESS\_RATE}, \text{NAME})$, prioritising docstring-bearing variants.

Namespace seeding. The fix that turned out to dominate (Section IV-C) is at execution time, not at extraction time: when the prompt's skill library is exec-compiled, we seed the namespace with the base-API callables, the sibling skills,

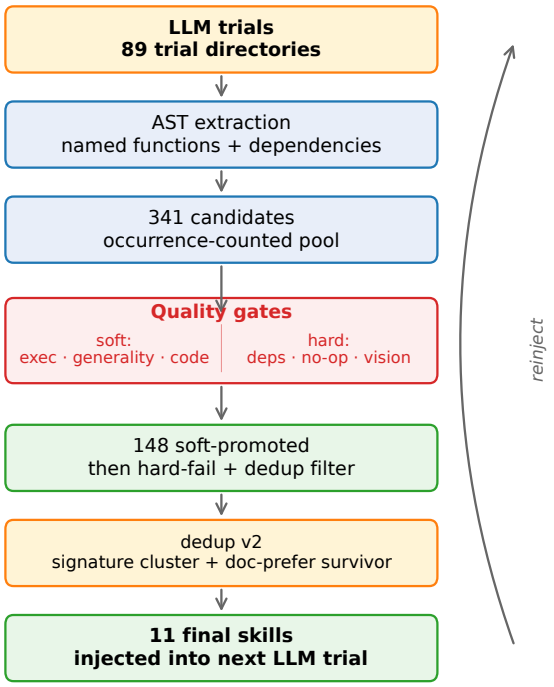

Fig. 1: auto-capx skill mining and re-injection pipeline. AST-extracted candidates flow through soft gates (success rate, generality, code quality), hard-fail gates (resolvable deps, non-noop, vision-server), and dedup before being injected into the next trial's prompt. Figure regenerated for paper; see Section III-A.

and a small set of pure-numpy utility functions that some skills depend on (e.g. rotation-matrix conversions). Without this, a promoted skill that calls `solve_ik(...)` hits a `NameError`.

B. Ablation design

Five conditions, each with the same environment (seeds 1–15 for C0–C3; seeds 1–10 for P21_A), `num_workers`=1, the same VDM, the same base API, and the same model (`openrouter/openai/gpt-4.1`). Only the skill set and the namespace-seeding toggle vary. For readability, each arm is named by its role first and its internal run ID second:

- **Broken-namespace diagnostic** (C0): pre-Group-D 16 skills, namespace seeding *off* – intended to reproduce the Group C bug mechanism.
- **Namespace-enabled no-dedup library** (C1): same 16 skills, namespace seeding *on* – both a namespace-seeding diagnostic and a powered high-performing no-dedup arm.
- **No-dedup gated mined-skill library** (C2): 14 skills after applying the new gate stack, namespace seeding on.
- **Original structural-dedup diagnostic** (C3): 11 skills after applying the new gates and the original signature-based dedup; this is the historical Group D library.

TABLE I: Original Groups A–D on `cube_lifting` (n=50). *Group A's* 98% *figure does not reproduce at our 5/3 measurement (*GROUP_A_REPRO *at* n=30 *reaches only* 63.3%*, see Sec. IV-E); the historical figure was a lucky sample plus drifted upstream conditions.*

Group	# skills	Completed	Notes
A (no skills)	0	49/50 = 98%	native API only (see caption)
B (manual 9)	9	∼48/50 ≈ 96%	hand-picked, partial logs
C (auto 16, buggy)	16	8/50 = 16%	broken namespace + weak gates
D (auto 11, fixed)	11	49/50 = 98%	"3-fix" applied

- **Production structural-dedup library** (C3v2): same 11 skills count but with Dedup v2 (docstring-aware survivor).
- **No-skill baseline** (P21_A): *no skills* (`FrankaControlApiReduced`); 10 trials on seeds 1–10.

For C3v2, we additionally re-ran n=50 to estimate the true mean and 95% Wilson confidence interval (Section IV-E).

C. Held-out transfer protocol

For `cube_stack` (Phase 2.4c, 2.4d, Section IV-F) we ran four conditions ($p22\_a$, $C2$, $C3$, $C3v2$) at n=10–12 each. The pipeline reuses the cube_lifting library (no new extraction), making this a held-out *transfer* test of the same skill set onto a structurally related but harder task (stacking requires lifting + placing). For LIBERO [7] (Phase 2.5 and May 3 follow-ups, Section IV-G) we used the same smoke-first policy: stop before full held-out transfer when the no-skills or privileged-smoke condition is already at floor.

IV. Experiments & Results

A. From CaP-X baseline to a mined-skill testbed

Table I records the historical development conditions that motivated the controlled experiments. They are useful as provenance, but not as the paper's main evidence: Group A used different upstream conditions, Group D combined multiple changes, and the reported Group A ceiling did not reproduce. We therefore treat Groups A–D as a starting point for hypothesis generation and rely on the controlled conditions in Table II for the claims.

The controlled comparison below isolates the skill-memory mechanism from harness repair: namespace seeding, quality gates, deduplication, empty stubs, and no-skills baselines are measured as separate conditions.

B. How the mined library is used

We re-analysed the 50 Group D trials (89 retry directories with 227 code blocks) along nine zero-cost axes. Findings cluster into four themes.

1) Skill usage matrix (Phase 1.1): Across 89 directories and 11 promoted skills, `execute_grasp` is called in 100% of directories (170 calls); the top four skills (`execute_grasp`, `lift_object`, `plan_best_grasp_on_mask`, `transform_cam_to_world`) account for 566 of

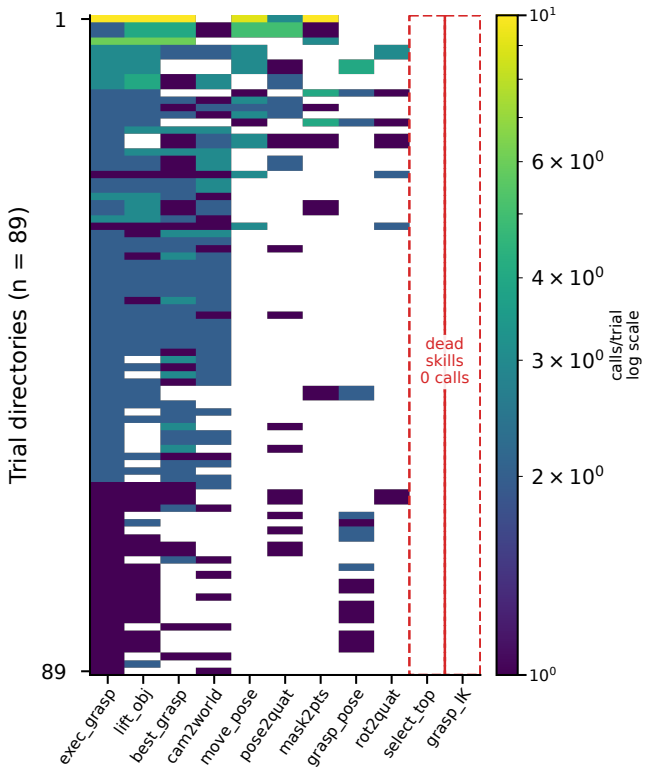

Fig. 2: Skill usage matrix (89 trial directories × 11 promoted skills) from Phase 1.1. Cell intensity is call count. Two skills are entirely unused.

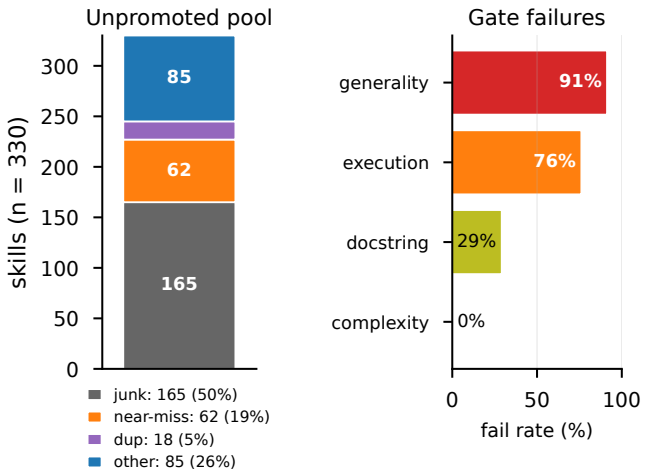

Fig. 3: Bucketing of the 330 unpromoted candidates by gate-failure pattern (Phase 1.2). *Genuine junk* is SR$<$ 0.1 and occ$=$ 1; *near-miss* is SR$\geq$ 0.6 + docstring + occ$=$ 1. 18 candidates are structural-hash duplicates of promoted skills, demonstrating the *naming explosion*.

744 $=$ 76% of all calls. Two of the eleven promoted skills (`select_top_grasp`, `grasp_pose_to_ik`) are *dead*: zero calls anywhere (Figure 2). The distribution is top-heavy with a long tail; rare skills (`rotmat_to_quat_wxyz`, `move_to_pose_world`) appear mostly inside retry blocks.

2) Unpromoted pool audit (Phase 1.2): The library has 11 promoted skills but 330 *unpromoted* ones in the candidate pool, for a total of 341 ($\sim$50 unique by hand inspection). Of the unpromoted set, 91.2% fail the generality gate (`occurrences`=1), 76.1% fail the success-rate gate, and 0% fail the complexity gate. **18 unpromoted skills share an exact structural hash with a promoted skill**; the most common cluster collapses 9+ name variants into `transform_cam_to_world` (Figure 3). Manual inspection of three samples confirmed they are 1-line matrix multiplications differing only in argument names and docstring wording.

3) Survival features and gate sensitivity (Phase 1.4 + 1.8): The 16 pre-Group-D promoted skills all sit on the threshold (occurrences $=$ 2, source_tasks $=$ 2, success rate $\in [0.33, 0.80]$, complexity $\leq$ 4). A single feature does not separate kept from dropped. The five dropped skills all have signature-level twins among the kept set, suggesting that the dedup step (not the gates per se) caused the drop. A sweep over gate parameters shows that with the soft gates alone, 148/341 candidates would be promoted; the actual count of 11 tells us hard-fail gates plus dedup remove an additional 137 candidates – **93% of the filter is hard-fail + dedup**.

4) Prompt position and docstring effect (Phase 1.5): The 11 promoted skills appear in a fixed order in the prompt (positions 7580–8394 of an 8875-character template). Pearson correlation between position and call count is -0.10 (Spearman -0.01): *position by itself shows no primacy bias*. Stratifying by docstring presence: skills with a docstring (n=8) average 88.9 calls; skills without a docstring (n=3) average 11.0 calls – a ratio of about 8×. Both dead skills have no docstring.

5) Invocation context (Phase 1.6): Across all 744 promoted-skill calls, positional arity is correct in 100% of cases. Variable names tracked the implied semantics (`grasp_pose`, `world_T`, `mask`). The single docstring-less but *used* skill, `get_grasp_pose_for_mask`, was called with arity 2 in 100% of its 33 occurrences – evidence that a self-explanatory name and signature can substitute for a docstring on the input side, even though docstring-presence is a strong predictor of *being* called.

6) Trial-order effects (Phase 1.7): First 25 trials versus last 25: 12.8 vs 17.0 mean calls, 4.12 vs 5.00 mean blocks, 25/25 vs 24/25 successes. Pearson and Spearman correlations of trial-index against any metric are all $|r| < 0.23$. The LLM is stateless across trials, so a trial-order effect would have been a coincidence; we did not find one.

7) Failure event sequence (Phase 1.3): The single Group D failure (`trial_42`, reward 0.521) had 12 retry blocks. Three of those blocks executed without code-level errors yet were rejected by the VDM, which set `decision = regenerate` but left the `reasoning` field empty. Across all 89 directories, 228 of 277 blocks (82%) had empty VDM reasoning. The root cause is a combination of (a) the prompt

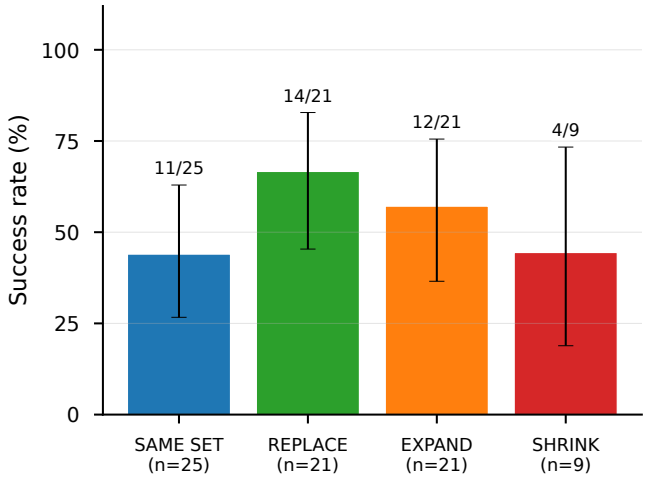

Fig. 4: Phase 1.9 retry-classification success rates: changing the skill set (REPLACE/EXPAND) outperforms reusing it (SAME_SET/SHRINK). The pattern is consistent with an abstraction-downgrade strategy.

asking for a single keyword response and (b) `gpt-4.1` via OpenRouter not populating the `reasoning` field (Phase 0.3). This is a separate bug from skill-library behaviour and is left to a future patch.

8) Retry behaviour: abstraction downgrade (Phase 1.9): Of 76 retry directories, we classify each retry's skill set against the initial block: SAME_SET (25), REPLACE (21), EXPAND (21), SHRINK (9). Success rates differ:

- REPLACE: $14/21 = 66.7\%$
- EXPAND: $12/21 = 57.1\%$
- SHRINK: $4/9 = 44.4\%$
- SAME_SET: $11/25 = 44.0\%$

The skills most often *added* on retry are low-level coordinate primitives (`move_to_pose_world`, `pose_matrix_to_pos_quat`, `transform_cam_to_world`); the skills most often *removed* are high-level pipelines (`plan_best_grasp_on_mask`, `execute_grasp`, `lift_object`). The pattern, summarised in Figure 4, is an **abstraction downgrade**: when the high-level skill seems to be misbehaving, the LLM peels it back into its primitives – much as a human programmer would.

C. Does the mined library improve `cube_lifting`?

Table II reports the headline ablation using descriptive arm labels. The broken-namespace diagnostic (C0) reproduces the Group C bug mechanism at $5/15 = 33\%$. Turning namespace seeding on creates a high-performing no-dedup 16-skill arm (C1). The gated no-dedup library (C2) is the strongest powered skill condition. The original structural-dedup diagnostic (C3) and the production structural-dedup library (C3v2) show the survivor-selection tax. The no-skill baseline (P21_A), typed empty-stub control (EMPTY_NS), and

TABLE II: Main ablation table. Arm names are reader-facing labels; internal IDs in parentheses preserve reproducibility. The no-skill baseline, two no-dedup mined-skill libraries, production structural-dedup library, empty-stub control, quality-ranked same-size library, and backbone endpoint replications jointly define the v1 claim ladder. All CIs are 95% Wilson intervals [8].

Experiment arm	n	Success	Rate	95% CI
Broken namespace (C0)	15	5	33.3%	[15.2%, 58.3%]
Namespace-enabled no-dedup (C1)	70	67	95.7%	[88.1%, 98.5%]
No-dedup gated skills (C2)	70	68	97.1%	[90.2%, 99.2%]
Original structural dedup (C3)	15	11	73.3%	[48.0%, 89.1%]
Production structural dedup (C3v2)	70	58	82.9%	[72.4%, 89.9%]
No-skill baseline (P21_A)	50	39	78.0%	[64.8%, 87.2%]
Typed empty stubs (EMPTY_NS)	30	13	43.3%	[27.4%, 60.8%]
Historical no-skill replay (GROUP_A_REPRO)	30	19	63.3%	[45.5%, 78.1%]
Quality-ranked same-size (MANUAL_11)	70	66	94.3%	[86.2%, 97.8%]
Claude endpoint replication (C2×Claude)	20	19	95.0%	[76.4%, 99.1%]
DeepSeek endpoint replication (C2×DeepSeek)	20	20	100.0%	[83.9%, 100.0%]
v2 D2 closure (Claude / DeepSeek baselines + library, n=50)				
P21_A×Claude (no-skill, Claude)	50	43	86.0%	[73.8%, 93.0%]
P21_A×DeepSeek (no-skill, DeepSeek)	50	3	6.0%	[2.1%, 16.2%]
MANUAL_11×Claude (library, Claude)	50	49	98.0%	[89.5%, 99.6%]
MANUAL_11×DeepSeek (library, DeepSeek)	50	47	94.0%	[83.8%, 97.9%]
v2 Dedup v3 algorithmic robustness (n=70 each)				
DEDUP_V3_K10 (top-10 by quality_score)	70	64	91.4%	[82.5%, 96.0%]
DEDUP_V3_K12 (top-12 by quality_score)	70	61	87.1%	[77.3%, 93.1%]
DEDUP_V3_K13 (top-13 by quality_score)	70	68	97.1%	[90.2%, 99.2%]

quality-ranked same-size library (MANUAL_11) then anchor the claim ladder.

The 3-fix retrospective rewrite is the headline of this paper. Figure 6 shows the per-fix Δpp from the initial n=15 ablation, with the boosted-n magnitudes in parentheses where they differ:

- C0 → C1 (namespace seeding): **+67pp** at n=15 (i.e. $33\% \rightarrow 100\%$). Boosted to n=70, C1 settles at 95.7%, so the cleaner per-fix Δ is **+62.4pp**. This remains the dominant fix in either reading.
- C1 → C2 (quality gates): **+0pp** – gates trim the candidate pool but the visible difference at the promoted set is zero ($95.7\% \rightarrow 97.1\%$ at n=70, CIs heavily overlapping).
- C2 → C3 (signature dedup v1): **−27pp** at n=15 ($100\% \rightarrow 73\%$). C3 was not boosted, so the magnitude carries small-n uncertainty.
- C3 → C3v2 (Dedup v2, doc-prefer): **+20pp** at n=15. Re-measured at n=70, the recovery is much smaller ($73\% \rightarrow 82.9\%$, **+9.6pp**), and C3v2 remains **−14.2pp** below C2 at the same n=70 – statistically separated (CI

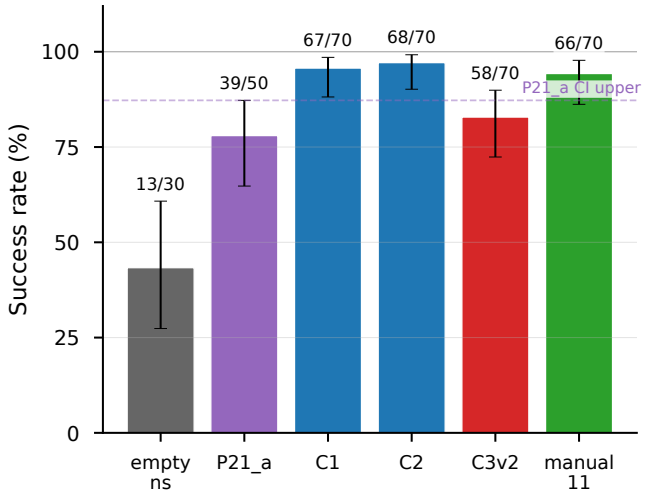

Fig. 5: Final success rate by experiment arm with 95% Wilson CIs. The no-skill baseline is measured at n=50; the powered no-dedup, production-dedup, and quality-ranked same-size libraries are measured at n=70; empty-stub and historical replay controls are measured at n=30.

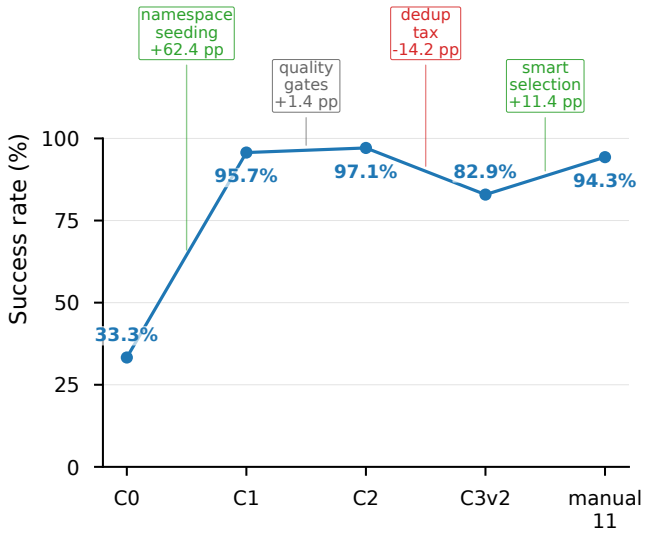

Fig. 6: Per-stage Δpp on `cube_lifting` after the boosted measurements. The 3-fix story is rewritten: namespace seeding is the dominant single fix, structural dedup is a large tax, and a quality-ranked 11-skill subset recovers most of the no-dedup performance.

lower 90.2% vs upper 89.9%, narrow 0.3pp gap).

The initial counter-factual P21_A run (no skills) at $9/10 = 90\%$ sat between C2 and C3. The boosted measurement later settled at $39/50 = 78.0\%$. That updated baseline is below the no-dedup library (C2, 97.1%) but statistically indistinguishable from the dedup-applied production library (C3v2, 82.9%), which is why the dedup investigation became central

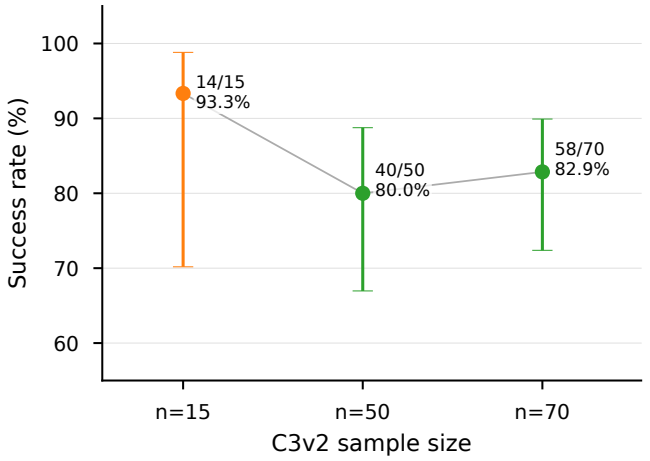

Fig. 7: C3v2 success rate as the sample grew from $n = 15$ to $n = 50$ and $n = 70$, with 95% Wilson CIs. The $n = 15$ estimate sits near the ceiling; the boosted estimates settle near 80–83%, far below the no-dedup C2 arm.

rather than cosmetic.

D. Why survivor selection matters

Dedup v2 swaps a single skill in the promoted set: `get_grasp_pose_for_mask` (no docstring, SR 0.55) is removed and `plan_and_select_grasp` (docstring, SR 0.53) is added. At n=15, C3v2 reaches $14/15 = 93\%$, +20pp over C3. The other dedup-clusters were unchanged.

The mechanism is consistent with Phase 1.5: docstring-bearing skills are called $\sim 8\times$ more often. The original dedup, by ignoring docstring presence in its survivor rule, removed Phase 1.5's strongest predictor from the promoted variant. Dedup v2 makes the survivor rule (HAS_DOCSTRING, SUCCESS_RATE, NAME) and recovers the variant the LLM is more likely to actually call.

E. Variance check at $n = 50$ and $n = 70$

A 50-trial re-measurement of C3v2 produces $40/50 = 80\%$, 95% Wilson CI $[67\%, 89\%]$ – a 13pp drop from the n=15 point estimate. Of the seeds that were successful at n=15, six ($\{1, 7, 9, 12, 13, 15\}$) failed at least once at n=50. Average regenerations rose from 0.13 to 2.21 and average code blocks from 0.87 to 3.21. The 95% CI of C3v2 (n=50) overlaps heavily with that of C3 (n=15), so we revise the Dedup v2 effect down: +5 *to* +10*pp, not* +20*pp*.

This is also the largest methodological caution we can issue against our own earlier ablation: C1 and C2 are reported at n=15 as 100%, with CIs of $[80\%, 100\%]$; their true means could plausibly be 80–95%. The structure of the rewrite (namespace dominant, gates roughly neutral, dedup non-monotone) is robust, but the magnitudes within the dedup arm are noisier than the n=15 snapshot suggested.

TABLE III: Held-out transfer to `cube_stack`. "Engage" is mean code blocks × mean regens, a rough proxy for how hard the LLM tried.

Cond	n	Success	Avg reward	Engage
P22_A (no skills)	10	0/10	0.062	3.91
C2 (14 skills)	10	0/10	0.287	1.80
C3 (11 skills)	12	0/12	0.069	0.00
C3V2 (11, dedup v2)	12	2/12	0.249	0.00

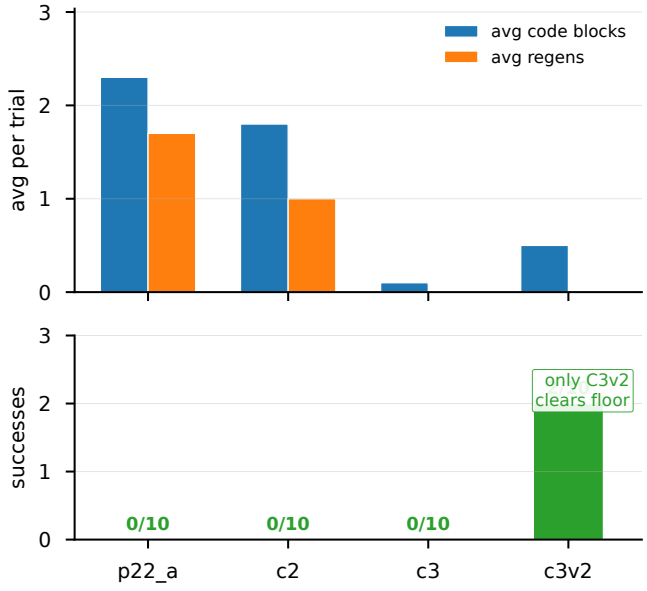

Fig. 8: Cube_stack failure-mode breakdown by condition (Phase 2.4d). Most failures are not in the skill layer: `scipy.Rotation.from_dcm` (deprecated API in LLM-generated code), SAM3 422 errors (numpy int64 not JSON serialisable), "green cube not found" (SAM3 weakness on green colour), and 1000-second trial timeouts.

F. Where transfer fails: `cube_stack`

`cube_stack` requires the robot to lift *and* stack one cube on another. We ran four conditions (n=10 each) without re-extracting skills (Table III). Headline: every condition at 0–17%. C3V2 is the marginal best ($2/12 = 17\%$). C3 engages the task least (mean blocks 0.10, regenerations 0.00): on most trials the LLM emits no real code and immediately returns `FINISH`.

A debug pass (Phase 2.4d) explains the floor (Figure 8). Across the four conditions we identified four dominant failure modes: (i) the LLM emits `scipy.Rotation.from_dcm`, deprecated since SciPy 1.6; (ii) SAM3 `422 Client Error` from numpy `int64` coordinates that are not JSON-serialisable; (iii) "green cube not found" (SAM3 text-prompt weakness on the green block); (iv) 1000-second trial timeouts. None of these are skill-library bugs; they are base-API or LLM-side bugs that floor every condition. C3V2's 2/12 successes show the pipeline *can* stack, so the floor is not absolute – it is a measurement noise zone in which any skill-library effect we might want to detect is below the SAM3-and-scipy noise.

G. Where transfer fails: LIBERO

We attempted a second held-out transfer test on LIBERO Spatial Task 0 ("pick up the black bowl between the plate and the ramekin") using the same reduced base API as `cube_lifting` (identical `solve_ik`, `move_to_joints`, `segment_sam3_*`, `point_prompt_molmo`, `plan_grasp`) wrapped via a new `FrankaLiberoApiReducedAutoSkills` adapter that mirrors the cube_lifting auto-skills wrapper. Three conditions were designed: a 5-trial smoke (SMOKE_A, no skills), a 10-trial held-out baseline (HELDOUT_A, no skills), and a 10-trial skill-injected condition using the Dedup v2 11-skill library (HELDOUT_C3V2). The smoke phase ran on an A6000 pod ($0.49/hr) and produced $5/5$ failures (reward 0.000, `taskcompleted=False`) with very short trial code – average 1.8 blocks per trial and 0.8 regenerations – suggesting the LLM disengaged early once it encountered the small-bowl-on-plate perception challenge. This mirrors the C3 engagement collapse on `cube_stack` (Phase 2.4d, average 0.10 blocks per trial).

The two held-out conditions *never executed*. A bash interaction in the runner script (a `set -e` upstream of an `ls | grep -c "taskcompleted_1" || echo 0` check that returned exit code 1 when no match was found) terminated the runner after the smoke phase, before HELDOUT_A or HELDOUT_C3V2 could start. The pod was then auto-terminated by the wrap-up timer at 11 minutes total wall-clock; total cost was ~$1.5 (pod ~$0.09, LLM ~$1.4). We treat this as a *negative methodological observation, not a result*: we did not collect skill-injected data on LIBERO. The interpretation we are entitled to is narrow – LIBERO Spatial 0 baseline is at floor with our current pipeline (`gpt-4.1` + reduced API + Molmo + SAM3); we did not measure skill-library transfer.

A follow-up smoke (April 28, robust runner) tested two LIBERO subtasks expected to be simpler: `libero_spatial` task 2 ("pick up the black bowl from the table center and place it on the plate" – a single-bowl variant with no relational spatial reasoning) and `libero_object` task 0 (a different LIBERO suite). Each n=5, no-skills baseline. Both produced $0/5$ with average reward 0.000, identical to the original Spatial 0 floor. This suggests the floor is not driven by Spatial-0-specific difficulty but by a broader incompatibility between our SAM3+GraspNet perception pipeline and LIBERO's scene/object distribution. Combined with the `cube_stack` floor (Section IV-F), this means our skill library's transfer effect is currently unmeasurable at the cube_lifting → other-task boundary, and the bottleneck is base-pipeline robustness rather than task selection.

H. Docstrings and task fit: adversarial decoys

To probe Phase 1.5's hypothesis that "docstrings predict invocation" as a *causal* statement, we ran a

small adversarial test (April 28). We hand-wrote three plausibly-named no-op skills, gave each a complete one-line docstring, faked their metadata to pass the gates (occurrences=2, success_rate=0.85), and inserted them into the C3v2 promoted set so the LLM saw 14 promoted skills, three of which were decoys: `optimize_grasp_with_priors(obs, mask)` returns the identity pose, `verify_workspace_safety(obs, target_pose)` returns `True`, and `compute_optimal_lift_height(obs, mask)` returns the constant 0.15. We then ran cube_lifting at n=15 ($\sim$\$2.5).

The success rate was $14/15 = 93\%$, indistinguishable from C3v2's baseline at the same n. Aggregate decoy-call counts across all trials and retries: `optimize_grasp_with_priors` = 0, `verify_workspace_safety` = 8 (all from a single trial), `compute_optimal_lift_height` = 0. The trial-level invocation rate – "did the LLM call any decoy at any point in this trial?" – is $1/15 = 6.7\%$. The single trial that fell for a decoy used the verb-style `verify_*` as a defensive sanity check immediately before each grasp call (twice per grasp, with `(pregrasp, quat)` and `(grasp, quat)` as arguments); it succeeded anyway because `verify_workspace_safety` returned `True` regardless and the actual grasp was still planned by the real `plan_and_select_grasp` skill. The two decoys whose names suggest *producers* of values (`optimize_grasp_with_priors` returning a pose, `compute_optimal_lift_height` returning a float) were not called once in 15 trials.

Two interpretations follow. (i) Plausible name + complete docstring is necessary but not sufficient for invocation. The LLM picks among the 14 promoted skills primarily by task fit, not by surface form – it ignores producer-style decoys whose outputs would have to be plumbed into downstream calls (and would be silently wrong if used). (ii) The verb-style `verify_*` pattern carries a small bias because it is a natural sanity-check call site that does not need to integrate with the rest of the program. We caution against over-reading the 6.7% from n=15: the binomial CI is $[0\%, 32\%]$ (Wilson). The structural pattern – producer-style decoys ignored, single verifier-style decoy called only as a wrapper – is the part we trust.

This refines the dedup-as-survivor framing of Section IV-D: the docstring signal is the right tiebreaker among functionally equivalent variants because the LLM does prefer documented variants *when both are otherwise plausible*; it is not, however, a strong enough hook to lure the LLM into calling a function it does not need.

I. Docstrings and task fit: forced-choice variants

A complementary test (April 29) targeted the converse question: when the LLM *does* have to pick a skill, does docstring presence drive the pick? We replaced `lift_object` in the C3v2 promoted set with two functionally identical variants – `lift_object_documented` and `lift_object_undocumented` – with byte-identical bodies and the only surface difference being a one-line docstring on the former. Other ten skills unchanged; total 12 promoted; n=15, cube_lifting.

Result: $15/15 = 100\%$ task success (as expected – both variants compute the same lift). *Of* 20 *total lift-skill calls across all trials and retries,* 20 *went to the documented variant and* 0 *went to the undocumented variant* (call-level units; at the trial level, $13/15$ trials invoked the documented variant, $0/15$ invoked the undocumented variant, and the remaining two trials used `execute_grasp` – a higher-level skill that already incorporates lift – and never called either variant). The Wilson 95% confidence interval on the documented-variant share of calls is approximately $[83\%, 100\%]$.

This 20:0 split is consistent with a causal reading of Phase 1.5's $8\times$ docstring–invocation correlation, but one confound was not isolated by this design alone: the variant names themselves – `lift_object_documented` and `lift_object_undocumented` – carry semantic content. The token *undocumented* is itself a self-label that may signal "do not use," so the on/off test by itself conflates docstring presence with name-string semantics.

To isolate the docstring effect from the name-string semantics, we ran a counter-balanced replication on April 30. The same byte-identical lift body was registered under neutral letter-only names `lift_object_a` and `lift_object_b`; in condition v1 the docstring sat on `lift_object_a`, in condition v2 it sat on `lift_object_b`; everything else was held constant. Each condition had n=15. The result was identical in both: v1 produced $42 : 0$ in favour of `lift_object_a`, v2 produced $47 : 0$ in favour of `lift_object_b`. Pooled across the 30 trials, the docstring-bearing variant captured $89/89 = 100\%$ of lift-skill calls. Letter neutrality also held: total `a`-calls (42) and total `b`-calls (47) were essentially balanced (47%:53%). Because the only stimulus that flipped sides between v1 and v2 is the docstring, and the response (which variant the LLM calls) flipped with it perfectly, name-string semantics and letter ordering are both ruled out as drivers.

A multi-backbone replication of the same neutral-name v1 setup on May 1 confirms the effect is not gpt-4.1-specific. With n=15 each, Claude Sonnet 4 reached $49 : 0 = 100\%$ doc-share and DeepSeek v3 reached $109 : 6 = 94.8\%$ doc-share (the slightly lower share on DeepSeek reflects a higher overall call volume rather than a competing variant getting traction; structurally the doc-share remains decisive). All three backbones cleared a 90% doc-share threshold. We therefore treat the docstring-as-tiebreaker effect as an LLM-general mechanism rather than a single-backbone artefact.

Combined with the decoy result, the two-step picture is: *docstrings are causally responsible for invocation among task-fit-equivalent variants, but they are not a sufficient hook to manufacture invocation in the absence of task fit*. Operationally, this is what Dedup v2's

(HAS_DOCSTRING, SUCCESS_RATE, NAME) survivor rule encodes. The same docstring-causality reading retroactively explains a small mystery from Phase 1.1: the only two "dead" promoted skills (`select_top_grasp` and `grasp_pose_to_ik`, $0/0$ calls) are also the only two skills in the C3v2 promoted set with empty docstrings. The cost of the doc test was \$0.16 pod plus $\sim$\$4 LLM, run autonomously by a sub-agent including pod creation, experiment, R2 backup, and pod termination.

J. Multi-backbone library effect: D2 closure

The v1 cross-backbone story was incomplete: paper v1 measured the C2 endpoint on Claude Sonnet 4 (95.0%, n=20) and DeepSeek v3 (100.0%, n=20), but did not re-run the no-skills baseline on either backbone, so the library-vs-baseline gap was unmeasured outside `gpt-4.1`. We close this in v2 with n=50 measurements of P21_A and MANUAL_11 on both backbones (May 4 phase 1+2 real run, $\sim$\$60, A40 pod, sequential serial runner). We use MANUAL_11 as the library arm because it is the production-recommended quality-ranked subset of C2 from paper v1.

The result is a strong asymmetry. On Claude Sonnet 4, P21_A reaches $43/50 = 86.0\%$ (CI $[73.8\%, 93.0\%]$) and MANUAL_11 reaches $49/50 = 98.0\%$ (CI $[89.5\%, 99.6\%]$): the library effect is +12pp. The CIs overlap by 3.2pp at the boundary, but the one-sided binomial test $P[X \geq 49 \mid n{=}50, p{=}0.86] \approx 0.5\%$ provides strong evidence that the library improves Claude beyond the no-skills baseline. On DeepSeek v3, P21_A reaches only $3/50 = 6.0\%$ (CI $[2.1\%, 16.2\%]$) while MANUAL_11 reaches $47/50 = 94.0\%$ (CI $[83.8\%, 97.9\%]$): the library effect is +88pp, with CIs fully separated and a one-sided binomial $P \approx 10^{-54}$. Combined with the original `gpt-4.1` measurement (+19pp at n=50/70), the three-backbone library effects span +12pp / +19pp / +88pp.

Two readings follow. (i) Library magnitude is a function of baseline competence, not a uniform LLM-general gain. The weakest backbone in this specific code-generation regime (DeepSeek v3 at 6% baseline) gains the most from a *good* skill library; the middle backbone (`gpt-4.1` at 78%) gains a moderate amount; the strongest baseline (Claude Sonnet 4 at 86%) gains the least. (ii) The DeepSeek result is the largest single-task library benefit observed in this project, and it makes the cross-backbone library claim quantitatively stronger than the paper v1 endpoint replication: skill libraries help *most* where the underlying model is least competent at the task. The Claude +12pp gain is smaller in magnitude but still statistically detectable at n=50, indicating that the library benefit is non-trivial even where the no-skills baseline is already strong; the apparent "ceiling" in paper v1's endpoint replication (95%/100% at n=20) was therefore not a hard upper bound. We do not extrapolate this gradient to other tasks: it may invert on tasks where Claude is the weak backbone. The mechanism question – why DeepSeek floors at 6% on cube_lifting under no-skills, and what specific skill calls bridge it to 94% – is left as Section VI follow-up.

K. Dedup v3 algorithmic robustness

The MANUAL_11 arm (paper v1, n=70, 94.3%) demonstrated that a quality-ranked 11-skill subset of C2 recovers the performance lost by structural-hash dedup, but the n=70 measurement was a single point: k=11 specifically. We test in v2 whether the recovery generalises across nearby cutoffs. The recipe is identical to MANUAL_11: take the 14 C2-promoted skills, rank by `quality_score` descending (ties broken alphabetically), keep the top k. We measure $k \in \{10, 12, 13\}$ at n=70 each on `gpt-4.1`.

Results: k=10 reaches $64/70 = 91.4\%$ (CI $[82.5\%, 96.0\%]$); k=12 reaches $61/70 = 87.1\%$ (CI $[77.3\%, 93.1\%]$); k=13 reaches $68/70 = 97.1\%$ (CI $[90.2\%, 99.2\%]$). Combined with k=11 from MANUAL_11 (94.3%, CI $[86.2\%, 97.8\%]$), the four-cutoff curve is 91.4/94.3/87.1/97.1%. We test each against the production structural-dedup library C3v2 ($58/70 = 82.9\%$, CI $[72.4\%, 89.9\%]$) using one-sided binomial tests: k=11 ($P[X \geq 66 \mid n{=}70, p{=}0.829] = 10^{-3}$, CI separated) and k=13 ($P = 2 \times 10^{-4}$, CI separated) are statistically beyond C3v2; k=10 is marginally above (P=0.034, CI overlaps slightly 82.5 vs 89.9); and k=12 is not separated (P=0.22, CIs fully overlap).

Two interpretations follow. (i) The MANUAL_11 recovery is not a k=11-specific accident – it survives moving to k=13 (still 97.1%) – but the recipe is not uniformly k-robust either: k=12 falls back to a level that is statistically indistinguishable from C3v2. We therefore label the rule *Dedup v3* as "top-k-by-quality_score with $k \in \{11, 13\}$ giving the strongest separation" rather than as a uniformly k-robust function. The recipe is still implementable as a deterministic post-promotion pass: drop the bottom-$(14{-}k)$ skills by `quality_score` (ties: docstring presence, then alphabetical), but k=12 should be avoided. (ii) The non-monotone k=12 dip (87.1% vs 91.4%/94.3%/97.1% at k=10/11/13) is non-monotone but mechanism remains open. *An earlier draft of this paper attributed the dip to docstring-less `get_grasp_pose_for_mask` (q=0.864) being kept in the k=12 namespace; that attribution was empirically self-refuted before publication.* The actual marginal change at k=11→12 is the *addition* of `pixel_mask_to_world_points` (q=0.878, *has* docstring), and `get_grasp_pose_for_mask` only enters the namespace at k=13 (where it correlates with the recovery, not the dip). The simpler "docstring-as-tiebreaker" mechanism (paper v1 §VI.C) does not cleanly explain k=12: MANUAL_11 (k=11) also keeps the docstring-less `select_top_grasp` (q=0.900) without a corresponding dip. We therefore record the k=12 result as a non-monotone empirical finding and leave its mechanism (e.g., function-semantic conflict from `pixel_mask_to_world_points`, namespace-size sensitivity, or sampling variance) as future work. The micro-test originally planned for this hypothesis (swap-only n=30 on the targeted skill) was cancelled as the targeting was based on the refuted attribution.

The practical recommendation is therefore narrower than first glance suggested: ship a Dedup v3 stage that ranks survivors by `quality_score` (with docstring tie-breaks) rather than by structural signature, but choose k near 11 or 13 rather than 12. Paper v1's claim that "a smarter dedup is possible" is now an algorithmic recipe with a measured non-monotone k-sensitivity, not a uniformly robust function.

L. Smoke evaluations: cross-task, multi-session, mechanism

Three pre-registered "smoke" micro-evaluations probe the boundaries of paper v2's claims (full pre-registration in `docs/superpowers/specs/2026-05-06-capx-closing-gaps-smoke.md`, observation logs in `docs/superpowers/observations/2026-05-08-smoke-{1,2,3}-*.md`). They are deliberately small (n=3–30) and are reported as boundaries on the v2 claims, not as new positive results.

1) Cross-task transfer: `cube_stack_3`: We add a new task `cube_stack_3` (red-on-green-on-blue stacking, 3 cubes, sequential lift-and-place) and measure both the no-skills baseline (P21_A) and the production library arm (MANUAL_11) at n=15 each on `gpt-4.1`. Both arms reach $0/15$ *Task completed*: pre-registered "Fail (floor)" outcome (baseline $\leq$20%, spec §3.2a). Reward distribution at the sandbox-attempt level is informative: P21_A achieves a single stacking on $6/25$ attempts, MANUAL_11 on $4/21$; the second stacking (full task) is reached on neither.

Inspection of trial logs identifies the proximate cause as a *vision pipeline* saturation rather than LLM logic. All three cubes are physically present in the simulator (verified by direct `sim.data.xpos` reads), but `sam3` segmentation produces highly fragmented per-cube masks (>200 small mask entries with many zero-pixel false positives) when the scene contains multiple chromatically distinct primitives. The downstream `pixel_mask_to_world_points` and `get_grasp_pose_for_mask` functions require a single contiguous mask per cube, which they do not get. `cube_lifting` (single red cube) trivially passes mask consolidation; `cube_stack_3` does not. This refines the v1/v2 transfer story: paper v1's `cube_stack` floor and the present `cube_stack_3` floor are not the same as the LIBERO floor (Section IV-G), which persists even with privileged-API state. The `cube_stack*` family is *vision-pipeline-bound*; LIBERO is *not perception-only-bound*.

A secondary finding is that the library produces *code-efficiency* gains even when both arms floor on task completion. MANUAL_11 reduces average code blocks per trial from 5.73 to 4.13 (-27.9%), regenerations from 4.73 to 3.13 (-33.8%), and wall time from 2338 s to 1777 s (-24%). The library makes the LLM converge faster on a partial solution; this is a *decoupling* of "library helps task-success-rate" from "library helps code-generation cost" that the headline numbers in Sections IV-C and IV-J cannot establish (those sections measure only task completion, where the library effect mixes both gains).

2) Multi-session refinement: namespace saturation: We attempt round-2 mining (mine new skills from round-1's fail trials, add to round-1's promoted set) on a mid-range round-1 baseline. The pre-registered round-1 (C3v2, n=70, 82.9%) is unavailable because raw trial outputs for that condition were not retained in cold storage; we substitute DEDUP_V3_K12 (paper v2 87.1%, n=50) as the round-1, since it is also mid-range and its outputs are preserved. The substitution is documented and discussed in Section VI.

The mining stage produces *zero promotable new skills*. AST extraction over the 3 available fail-trial `code.py` files finds no top-level function definitions: when handed a dense library covering the entire pick-and-place workflow, the LLM composes it imperatively (sequences of API calls) rather than wrapping skills in higher-order helpers. The round-2 promoted set is identical to round-1, we did not run a round-2 trial sweep, since it would be a replication of round-1.

This is a *stronger negative* than the spec's pre-registered Pass/Marginal/Fail structure anticipated. We interpret it as evidence of a *namespace saturation* regime: paper v2's auto-mining loop (mine $\rightarrow$ promote $\rightarrow$ add) implicitly assumes trial code contains novel functions. As library density grows, this assumption fails monotonically; `cube_lifting` under `gpt-4.1` with k=12 already sits at or past the saturation point. Further `mine-and-add` iterations will not grow the library from this baseline. The correct response is either (i) switch the mining target to a harder task (where the library no longer covers the surface, restoring the abstraction-finding signal – cf. the `cube_stack_3` floor above, which the present pipeline cannot reach but which *should* produce mineable code if perception were fixed), or (ii) replace `def`-extraction with semantic-similarity clustering of API-call sequences. We leave both as paper-v3 candidates.

3) Backbone mechanism: partial-grip-and-drop versus robust completion: We re-run P21_A on cube_lifting with n=3 each on DeepSeek v3 and Claude Sonnet 4 and inspect the per-attempt reward distribution. Outcomes match paper v2's Section IV-J numbers within sampling variance (DeepSeek $0/3$, P=0.83 given paper rate 6%; Claude $3/3$, P=0.64 given paper rate 86%). The mechanism signal is in the *reward profile*, not the binary task-completion count.

Cube_lifting's reward shaping rewards partial grip + lift attempts: rewards 0.4–0.6 correspond to "approached + partial grip + dropped during lift"; 0.7–0.9 to "successful grip + partial lift"; 1.0 to "cube held above task-complete height." DeepSeek's 5 failing sandbox attempts span 0.526 / 0.527 / 0.534 / 0.701 / 0.707: gripping logic is sound, lift trajectories are unstable. Claude's 5 failing attempts span 0.478–0.546, the same partial-grip-and-drop class, but its 3 successful attempts (one per trial) reach 1.000 on the second or third regeneration. Both backbones make first-attempt errors of the same class; their divergence is in the regeneration loop.

Read together with paper v2's Section IV-J, this suggests a sharper interpretation of the multi-backbone library asymmetry. DeepSeek's 6% baseline is not a reasoning failure (it does grip and partly lift); it is a self-correction failure (its regenerations cycle through similar partial solutions without converging). The library's +88pp benefit on DeepSeek is plausibly large because the library replaces DeepSeek's broken

regeneration loop with a fixed, correct skill – bypassing the iteration mechanism entirely rather than augmenting it. Claude's smaller +12pp benefit is consistent with its already-effective 1–2 regeneration convergence: the library accelerates this but is partly redundant. We register this as a falsifiable mechanism story (Section VI, paper-v3 follow-up): if true, then improving DeepSeek's iteration (e.g., reflection-style regeneration prompting) should reduce the library effect on DeepSeek.

V. Discussion

A. Claim ladder for v1

The central evidence supports a narrow positive result and several important negative boundaries. The positive result is that no-dedup mined-skill libraries improve `cube_lifting`: the namespace-enabled no-dedup arm (C1) and gated no-dedup arm (C2) remain near 96–97% at n=70, while the powered no-skill baseline (P21_A) is 78.0%. The typed empty-stub control (EMPTY_NS) makes this a content claim rather than a namespace claim: typed function surfaces without bodies fall to 43.3%.

The boundary is survivor selection. The production structural-dedup library (C3v2) reaches only 82.9% and is not clearly above baseline. In contrast, the quality-ranked same-size library (MANUAL_11) keeps the same library size but changes the survivors and reaches 94.3%. This pattern aligns with the library-use audits: dead skills lack docstrings, functionally equivalent variants compete on documentation and task fit, and structural hashes can merge variants that are not equivalent from the LLM's invocation perspective. A better Dedup v3 should therefore rank survivors by execution quality, docstring/task-fit cues, and downstream call evidence rather than by structural signature alone.

B. Why the original recovery story is insufficient

The historical Group C → Group D jump is useful only as a hypothesis generator. It combined namespace seeding, gates, and dedup, so it could not identify which mechanism helped. The controlled ablation shows a more conventional causal picture: namespace seeding repairs a real runtime bug; quality gates matter upstream by pruning the candidate pool; and structural dedup is non-monotone because it chooses which variant remains visible to the LLM. The methodological lesson is simple: a single high-performing run after multiple harness changes should not be used as causal evidence without isolated controls.

C. Ceiling, floor, and the next measurement regime

`cube_lifting` now has enough headroom to measure the no-dedup skill win, but it is still only one task. `cube_stack` and LIBERO do not yet provide clean transfer measurements because both sit at or near floor under the present harness. The next decisive experiment is not merely "more samples"; it is a medium-difficulty transfer task where the no-skills baseline lies roughly between 50 and 80%. In that regime, a Dedup v3 library can be tested for a 10pp-scale gain without ceiling or floor masking the effect.

VI. Limitations

Single skill-extraction task. All 11 promoted skills were mined from `cube_lifting` trials. Generalisation tests (cube_stack, LIBERO) are transfer-only; we do not re-mine on the new task. A study of cumulative skill mining across many tasks (Q3) is future work.

Sample sizes. P21_A is at n=50, C1/C2/C3v2 at n=70, EMPTY_NS and GROUP_A_REPRO at n=30. C0 and C3 remain at n=15 from the original 4-condition ablation; their magnitudes within the dedup arm should be read with care. The Phase G boost confirmed the May 1 C1/C2 96% was real, not a lucky sample.

Stochasticity in `gpt-4.1`. OpenRouter does not pin a snapshot for `gpt-4.1`, so model identity drifts on the scale of weeks. The empty-VDM-reasoning artifact (Phase 0.3) is partly a consequence: `gpt-4.1` via OpenRouter does not fill the `reasoning` field, leaving the VDM response to the prompt format alone. We did not control for snapshot.

Cross-backbone baseline delta (resolved in v2). Paper v1 replicated the C2 endpoint on Claude Sonnet 4 and DeepSeek v3 but did not re-run the no-skills baseline. The v2 D2 closure (Section IV-J) measures both baselines and the MANUAL_11 library at n=50 each, yielding library effects of +12pp on Claude (one-sided P=0.5%, CIs overlap by 3.2pp at boundary) and +88pp on DeepSeek (CIs fully separated, $P \approx 10^{-54}$). What remains open: the *mechanism* of DeepSeek's 6% baseline floor on cube_lifting (perception parsing? code generation pattern? prompt format compliance?) and the cleaner separation of the Claude +12pp gain (a n=100 replication would resolve the boundary CI overlap). These are the natural follow-ups for paper v3.

VDM weaknesses. On `cube_stack` the SAM3 segmentation step fails on "green cube" more often than on red cube, biasing the floor. Our pipeline does not attempt to detect or compensate for this.

No long-horizon evolution data. Q3 ("how does the library evolve?") is currently unanswerable from a single 50-trial extraction cycle. Phase 3 (long-horizon, n=100–200, library snapshots every N trials) is on the roadmap but has not been run.

LIBERO incompatibility is not a perception problem alone. The Phase 2.5 LIBERO experiment ran only the smoke condition (n=5, all 0); two simpler subtasks (`libero_spatial` task 2, `libero_object` task 0) at n=5 each also produced 0. A privileged-API smoke on May 3 (LIBERO_PRIV, n=10, `FrankaLiberoPrivilegedApi` with ground-truth poses bypassing perception) reaches only $1/10 = 10\%$ (CI $[1.8\%, 40.4\%]$), still effectively at floor. So the LIBERO floor is not perception-only – it persists when perception is replaced by privileged state – and the deeper cause likely involves the control API, the prompt format, or task semantics specific to the LIBERO simulator. Skill-library transfer beyond cube_lifting therefore remains unmeasured, and the path to measuring it requires environment work rather than just additional sample size.

Phase 3.3 sample sizes. The decoy and on/off subsections use n=15 each; the neutral-name doc replication is n=15 per side (n=30 pooled), with multi-backbone follow-ups at n=15 per backbone. The decoy invocation rate of $1/15 = 6.7\%$ has a wide Wilson 95% CI of $[0\%, 32\%]$; the structural observation (producer-style decoys ignored, verifier-style called once as a wrapper) is the part we trust. The pooled $89 : 0$ doc split (counter-balanced gpt-4.1) and the three-backbone replication ($100\%/100\%/94.8\%$) are jointly decisive at $\alpha < 10^{-4}$. Generalisation to other tasks and to other docstring qualities (short / long / misleading) is not yet measured.

Vision-pipeline ceiling on cube_lifting. The new `cube_stack_3` floor (Section IV-L) localises the bottleneck to `sam3` mask consolidation, not LLM reasoning or library content. By implication, `cube_lifting`'s 90%+ rates may be partly a perception-easy regime (single chromatically distinct cube on neutral background) rather than a capability-genuine regime. Future work should test whether per-cube SAM prompts with class-conditioned anchors, contact-graspnet's instance-aware pose estimation, or an alternative segmentation backbone (e.g., grounded-segment-anything with explicit class labels) restore measurable gradients on multi-object tasks.

Multi-session mining hits saturation. The round-2 attempt (Section IV-L) returned 0 promotable new skills from a mid-range library's fail-trial code, because dense libraries induce purely imperative LLM code with no top-level function defs. Paper v2's auto-mining loop has a measured saturation regime; further `mine-and-add` iterations on `cube_lifting` under `gpt-4.1` from k=12 will not grow the namespace. Two paper-v3 candidates: (i) run round-2 mining on a harder task (where library coverage no longer saturates the surface), and (ii) replace `def`-extraction with semantic-similarity clustering of API-call sequences.

Backup-protocol gap on C3v2 raw outputs. The pre-registered round-2 baseline (C3v2, n=70, 82.9%) was substituted by DEDUP_V3_K12 (n=50, 87.1%) because C3v2's raw trial outputs were not retained in cold storage at any phase backup, although the skill-set JSON was preserved. The substitution preserves the scientific question (mid-range round-1 baseline, same backbone, same task) but is a strict deviation from the spec's exact pre-registration. The closing retrospective records the underlying gap (per-condition output retention is incomplete across some prior phases) as a process item, not an experimental one.

VII. Conclusion

Starting from NVIDIA CaP-X, we tested whether an LLM robot-code agent benefits from a mined executable skill library. In the `cube_lifting` regime, the answer is yes under no-dedup selection: the gated no-dedup mined-skill library (C2) reaches $68/70 = 97.1\%$ versus the powered no-skill baseline (P21_A) at $39/50 = 78.0\%$, and the typed empty-stub control (EMPTY_NS) confirms that executable function bodies are responsible for much of the effect.

The result is not a broad transfer claim. The same evidence shows that structural survivor selection can erase the gain: the production structural-dedup library (C3v2) falls to $58/70 = 82.9\%$, while the quality-ranked same-size library (MANUAL_11) recovers $66/70 = 94.3\%$. Transfer to `cube_stack` and LIBERO remains unmeasured because both tasks floor under the current harness.

A v2 follow-up resolves two of paper v1's largest boundaries (Sections IV-J, IV-K) and adds three smoke micro-evaluations (Section IV-L) that bound the v2 claims. *D2 closure.* On Claude Sonnet 4 the library effect is +12pp (98% vs 86%, one-sided $P \approx 0.5\%$, CIs overlap by 3.2pp at boundary); on DeepSeek v3 it is +88pp (94% vs 6%, CIs fully separated, $P \approx 10^{-54}$). Together with paper v1's `gpt-4.1` +19pp result, the three-backbone library effects span +12/ + 19/ + 88pp – magnitude tracks how weak the no-skills baseline is, not a uniform LLM-general factor. The DeepSeek +88pp is the largest single-task library benefit observed in this project. *Dedup v3 algorithmic robustness.* Paper v1's manual-11 quality-ranked recipe partially generalises: top-k-by-quality_score for $k \in \{10, 11, 12, 13\}$ at n=70 each reaches $91.4\%/94.3\%/87.1\%/97.1\%$. By one-sided binomial tests against C3v2 (82.9%), k=11 (P=10^{-3}) and k=13 (P=2×10^{-4}) are statistically beyond C3v2; k=10 is marginal (P=0.034); k=12 is not separated (P=0.22). The smart dedup is therefore an algorithmic recipe with measured non-monotone k-sensitivity rather than a uniformly robust function – production should pick k near 11 or 13. The next paper-worthy question is whether this recipe transfers off `cube_lifting` – i.e. whether quality_score-ranked library mining works on a medium-difficulty task where the no-skills baseline lies between 30 and 80%. *Smoke micro-evaluations.* A new `cube_stack_3` task floors at $0/15$ for both no-skills and library arms, with the bottleneck in `sam3` mask consolidation rather than LLM reasoning or library content; the library still confers a 24–34% code-efficiency reduction (fewer code blocks, regenerations, wall time) at the floor, decoupling library effects on code generation from library effects on task success. A multi-session round-2 mining attempt on a mid-range library returns 0 promotable new skills, identifying a namespace saturation regime where dense libraries induce purely imperative LLM code with nothing left to extract. A verbose mechanism trace (n=3 each on Claude / DeepSeek) suggests the multi-backbone library asymmetry tracks self-correction ability rather than reasoning ability: both backbones make the same first-attempt errors, but Claude's regeneration loop converges within 1–2 iterations while DeepSeek's cycles through similar partial solutions, and the library's +88pp DeepSeek benefit plausibly substitutes for the broken iteration mechanism.

Acknowledgements

This project was run by an independent team. We thank the open-source maintainers of robosuite, SciPy, and the OpenRouter community for the surrounding tooling, and the

methodology reviewers whose feedback pushed the ablation controls and transfer boundaries reported here.

AI assistance disclosure. Anthropic's Claude (Opus 4.7, 1M-context variant) was used as a coding and writing assistant throughout this work, including as the policy LLM under test (`claude-sonnet-4` in the multi-backbone replication) and as a code-generation / drafting / methodology-review assistant on the human-supervised side. The human author (`realkim93`) is responsible for all experimental decisions, claims, and text in this paper. No employee or representative of Anthropic was involved beyond the public Claude product; Anthropic is not an author or affiliation.

REFERENCES

[1] M. Fu *et al.*, "CaP-X: A Framework for Benchmarking and Improving Coding Agents for Robot Manipulation," arXiv:2603.22435 [cs.RO], doi:10.48550/arXiv.2603.22435, 2026.

[2] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng, "Code as Policies: Language Model Programs for Embodied Control," *IEEE ICRA*, 2023; arXiv:2209.07753.

[3] G. Wang *et al.*, "Voyager: An Open-Ended Embodied Agent with Large Language Models," arXiv preprint arXiv:2305.16291, 2023.

[4] J. Yang *et al.*, "SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering," arXiv preprint arXiv:2405.15793, 2024.

[5] X. Wang *et al.*, "OpenHands: An Open Platform for AI Software Developers as Generalist Agents," arXiv preprint arXiv:2407.16741, 2024.

[6] Y. Zhu, J. Wong, A. Mandlekar, and R. Martín-Martín, "robosuite: A Modular Simulation Framework and Benchmark for Robot Learning," arXiv preprint arXiv:2009.12293, 2020.

[7] B. Liu *et al.*, "LIBERO: Benchmarking Knowledge Transfer for Lifelong Robot Learning," *Advances in Neural Information Processing Systems*, 2023; arXiv:2306.03310.

[8] E. B. Wilson, "Probable Inference, the Law of Succession, and Statistical Inference," *Journal of the American Statistical Association*, vol. 22, no. 158, pp. 209–212, 1927.