TRACE2PROG: COMPILING AND CALIBRATING AGENT EXPERIENCE INTO REUSABLE VISUAL PROGRAMS

Anonymous authors
Paper under double-blind review

ABSTRACT

In recurring visual question answering (QA), different questions may share the same solution procedure despite changes in their visual inputs. However, multimodal agents often reconstruct this procedure from scratch for every question. Reusing compiled procedures can reduce repeated processing, but a program that worked before may fail under new visual conditions. The key challenge is deciding when to use a compiled program and when to fall back to the agent. To this end, we introduce **Trace2Prog**, a compile–calibrate–deploy framework for recurring visual QA. It compiles past executions into candidate programs, validates them under the source visual condition, and uses a small labeled target set to determine whether they remain suitable for reuse. For each new question, Trace2Prog uses a validated program when suitable and otherwise falls back to the multimodal agent. On 480 controlled requests across four task types and three renderer profiles, Trace2Prog achieves 93.1% accuracy, compared with 48.3% for the baseline tool agent, 86.7% for per-query program synthesis, and 87.1% for success-only compilation. On 3,215 TIR-Bench and PuzzleVQA requests, Trace2Prog improves overall accuracy by 0.25–2.18 percentage points without introducing observed additional errors. Across 2,160 program–request pairs, target calibration reduces harmful replacements from 87 to 24.

1 INTRODUCTION

Tool-using multimodal agents couple a vision–language model (VLM) with tools that inspect images, execute code, and return an answer (Yao et al., 2023; Yang et al., 2023; Surís et al., 2023; Song et al., 2026). Their executions leave agent experience—tool calls, code, observations, and outcomes—that may encode reusable computations beyond the original request (Cai et al., 2024; Yuan et al., 2024). Repeated-symbol grids, for example, reuse segmentation and comparison, while shape-size puzzles reuse contour measurement. Reconstructing these procedures for every request repeats model work already present in earlier executions. We study how to compile that experience into persistent visual programs and when those programs should replace repeated agent execution.

Prior work either generates programs independently for each query or accumulates executable tools for later retrieval (Gupta & Kembhavi, 2023; Surís et al., 2023; Cai et al., 2024; Yuan et al., 2024; Ren et al., 2026; Jiang et al., 2026). These approaches show that procedures can be synthesized and reused, but compilation alone does not establish whether a stored program should replace an agent. A per-query error affects one request, whereas a brittle stored program may repeat the same error whenever fonts, layouts, colors, or task parameters change. A program that succeeds on historical executions may therefore be harmful under the target renderer. Deployment also cannot assume access to benchmark task labels. Reliable reuse must therefore separate three questions: can past executions yield a program that works on other requests of the same kind; does that program improve on the target agent under the intended visual conditions; and can it be applied without oracle task labels?

Figure 1 motivates three separate decisions. Source validation asks whether a program works beyond its construction traces. Target calibration asks whether replacing the agent corrects more answers than it breaks, measured by Fix, Hurt, and Net = Fix - Hurt. Deployment must then determine whether the program applies to the current request and retain the fallback under routing ambiguity or invalid execution.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

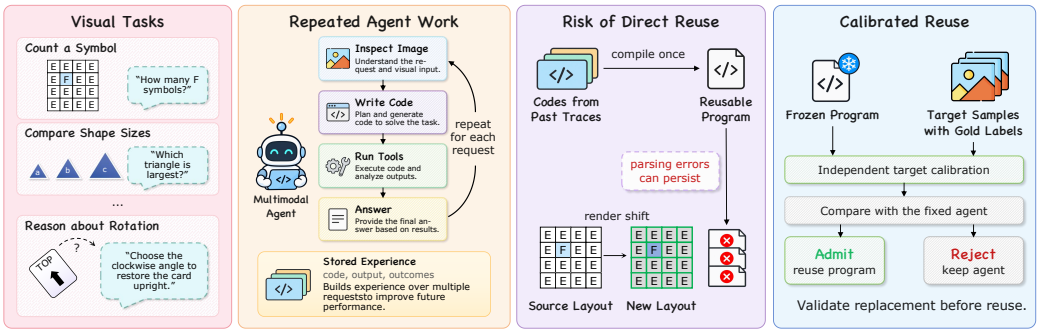


Figure 1: Why program reuse requires calibration. Similar visual requests induce repeated inspect–code–execute loops, allowing past traces to be compiled into a persistent program. Direct reuse can repeat parsing errors after a renderer shift. Trace2Prog therefore compares each frozen program with the fixed tool-agent fallback on labeled target samples before admitting it for reuse.

Trace2Prog implements these decisions through compilation, calibration, and deployment. A language-model compiler converts bounded executions into independent `solve(image, prompt)` candidates. Static checks, sandboxed replay, and held-out source validation select at most one program per observable task interface, or contract. Labeled target calibration admits the frozen program only when it meets a precision threshold and provides positive paired utility over the fixed fallback; calibration changes the admission decision, not the solver code. At inference, a label-free router sees the image and prompt but no contract label, deploying a program only when exactly one checked output remains. The solver runs locally without an additional solver-model call; routing uses one lightweight VLM call per request.

Our evaluation separates four data roles: source executions construct candidates, held-out source examples select a solver, labeled target examples calibrate admission, and final requests measure replacement utility. On 480 controlled final requests from four benchmark-derived contracts and three predeclared renderer profiles, Trace2Prog improves accuracy from 48.3% to 93.1%. In a separate retrospective analysis scored over the complete 3,215-request TIR-Bench and PuzzleVQA pool, four compiler registries deploy on 32–150 requests and add 8–70 correct answers with zero observed Hurt or wrong-contract deployment. Across 18 executable candidates, precision-plus-paired admission reduces the harms introduced by source-eligible programs from 87 to 24 while increasing Net from $+344$ to $+364$. Together, these studies test reusable-program construction, target replacement, and request-level application. Our contributions are:

- **Program reuse as conditional replacement.** We distinguish whether a program works on held-out source requests from whether it should replace a particular fallback under target conditions. By measuring Fix, Hurt, and Net on independent target data, this formulation makes both the benefits and risks of replacement explicit and exposes source-eligible programs with negative deployment utility.
- **A compile–calibrate–deploy framework.** Trace2Prog compiles bounded agent experience into standalone programs, validates them through sandboxed replay, and admits them using selective precision and paired utility. The controlled study freezes program bytes, admission decisions, and routing configuration before final evaluation; retrospective analyses are marked separately. Label-free routing and output checks support local deployment without task labels.
- **Evidence from controlled replacement and benchmark-wide scoring.** Complementary benchmark-pool, renderer-transfer, and candidate-level studies show selective gains, robustness under calibrated renderer changes, and fewer errors than source-only deployment. Threshold sweeps further expose risk–benefit operating points unavailable to precision alone.

2 Related Work

Executable skills and tool libraries. Reusable tool libraries amortize synthesis costs across requests. LATM separates tool construction from tool use (Cai et al., 2024), whereas CRAFT validates and abstracts task-specific code solutions into reusable tools (Yuan et al., 2024). TroVE instead induces compact, reusable toolboxes by generating functions through use and iteratively growing

and trimming the library (Wang et al., 2024). Extending tool reuse to continual agent adaptation, FlowEvo compiles successful workflows into executable skills and curates them according to their downstream utility (Ren et al., 2026). Recent multimodal systems further organize visually grounded experience and skills: XSkill supports continual accumulation and retrieval (Jiang et al., 2026), while Ace-Skill prioritizes rollouts and clusters evolving knowledge (Xiong et al., 2026). ContractSkill adds procedural structure, deterministic verification, and localized repair (Lu et al., 2026). Concurrent SkillSmith systems respectively co-evolve skills and tools from execution feedback (Wei et al., 2026b) and compile skill packages into boundary-constrained runtime interfaces (Xu et al., 2026).

Trace2Prog studies a complementary deployment question: after a complete visual solver has been compiled, should it replace a particular fallback under target rendering conditions? It freezes the solver and makes this decision using independent target calibration and paired replacement utility. This separates the usefulness of a stored skill from the benefit of replacing the agent that would otherwise answer the same request.

Experience and failure memory. Voyager accumulates executable skills through interaction (Wang et al., 2023), while ExpeL compares successful and failed trajectories to derive reusable insights (Zhao et al., 2024). WikiSkill consolidates execution experience into a persistent knowledge base (Tang et al., 2026), and the co-evolving SkillSmith explicitly records failure signatures, causal attributions, and remedies (Wei et al., 2026b). Trace2Prog similarly compiles prompts, images, outcomes, and execution evidence, but treats the composition of the source experience bank as an experimental variable. This allows experience selection to be studied separately from the decision to deploy the resulting program.

Visual programs and selective deployment. VisProg (Gupta & Kembhavi, 2023) and ViperGPT (Surís et al., 2023) synthesize programs separately for each visual query. Chameleon composes modular multimodal tools (Lu et al., 2023), and CodeDance studies dynamic, tool-integrated visual reasoning (Song et al., 2026). MetaForge retrieves, adapts, or creates tools on demand (Wei et al., 2026a), whereas MMSkills represents reusable multimodal procedures as state-conditioned packages (Zhang et al., 2026). Think–Verify–Revise instead induces visual rules and verifies them using differentiable logic (Afshari et al., 2026). Trace2Prog differs by paying the synthesis cost once, freezing the resulting solver, and executing it locally across subsequent requests.

Selective prediction studies abstention and risk–coverage trade-offs (Geifman & El-Yaniv, 2019). Trace2Prog adapts this perspective to a paired decision: whether an admitted solver improves over a fixed fallback on target-renderer calibration data.

3 Problem Formulation and Deployment Protocol

A reusable visual program can execute recurring operations such as symbol localization or contour measurement without reconstructing them for every request. Reuse nevertheless requires three distinct decisions: whether the program works across source requests, improves the target fallback, and applies to the current input.

3.1 From Agent Experience to Reusable Programs

We organize source experience by registered *contracts*: observable input–output interfaces that group construction episodes. Similar images may belong to different contracts when their answer schemas differ; contract identity is not supplied to the online router.

Each episode contains an image, prompt, agent answer, outcome, and available execution evidence. A bounded episode bank yields standalone `solve(image, prompt)` candidates that may answer, abstain, fail, or time out. Before replacement, an answer must pass a contract-specific interface checker; validity is distinct from correctness against the gold answer.

Held-out source validation evaluates frozen candidates on requests excluded from compilation. Candidates passing the source criteria are *source-eligible*, and selection retains at most one per contract for target evaluation. This tests cross-request usefulness without requiring the candidate to reproduce the exact algorithm used in a historical trace.

3.2 Paired Replacement Utility

Precision measures answer quality on requests a solver chooses to answer; replacement utility measures the change relative to the fallback on those same requests. A solver answering 90 of 100 requests correctly has 90% precision, yet replacing a fallback that answers 98 of those requests correctly loses eight correct answers. Conversely, positive utility over a weak fallback need not imply reliable program answers.

For a fixed solver s, let $S_s(D)$ denote the requests in D on which s returns a non-abstaining, checker-valid output. Support is $|S_s(D)|$, and selective precision is the fraction of outputs in $S_s(D)$ that match the gold answer. During contract-level evaluation, the solver replaces the fallback only on $S_s(D)$. Thus, selective precision depends on the returned solver answers, whereas replacement utility also depends on the designated fallback.

Let b_i denote the fixed tool agent's prediction, y_i the gold answer, and $\tilde{y}_i$ the prediction after conditional replacement on request i. A *Fix* corrects an agent error; a *Hurt* replaces a correct agent answer with an incorrect one. For an evaluation set D,

$$\Delta(D) = \frac{1}{|D|} \sum_{i \in D} [\mathbf{1}\{\tilde{y}_i = y_i\} - \mathbf{1}\{b_i = y_i\}] = \frac{\mathrm{Fix}(D) - \mathrm{Hurt}(D)}{|D|}. \tag{1}$$

The unnormalized $\mathrm{Net} = \mathrm{Fix} - \mathrm{Hurt}$ is the change in correct answers. Target admission requires both selective precision and paired utility for a specified fallback.

3.3 Admission and Request-Level Application

Target admission evaluates the source-selected solver on independent labeled examples from the intended deployment conditions. The decision binds the solver, checker, and fallback: changing the fallback changes Fix and Hurt and therefore requires recalibration.

Admission does not imply applicability to every request. Online routing uses the image and prompt to distinguish interfaces; for example, two programs may both return colors while operating on different geometric structures. Detectable ambiguity, invalid outputs, and execution failures retain the fallback. Because a format-valid answer can still be wrong, we separately measure recall, deployment coverage, wrong-contract deployment, and paired utility.

At deployment, let R_i be the admitted solvers recalled for request i. Replacement occurs only when routing confidence passes its threshold and exactly one solver in R_i returns a checker-valid output. Deployment coverage is the fraction of all requests that receive a program answer. Routing recall is the fraction of requests answerable under registered-contract execution that remain deployed after routing. Wrong-contract deployment counts deployed answers produced by a contract different from the evaluation-time contract; these contract labels are used only for evaluation and are unavailable online.

The protocol separates source experience, source validation, target calibration, and final evaluation. Candidate code is fixed before source validation and remains unchanged during calibration; admission decisions and the routing configuration are then fixed for final execution. The controlled study uses unseen requests from renderer profiles represented in calibration. Retrospective extensions instead reuse fixed evaluation requests and are identified in Section 5. Trace2Prog implements these decisions through compilation, target calibration, and task-label-free online deployment after labeled target calibration in Section 4.

4 Trace2Prog: Compile, Calibrate, and Deploy

Trace2Prog implements the decisions in Section 3 in three stages. As Figure 2 shows, compilation turns a bounded experience bank into candidate solvers and selects one on held-out source data. Target-side calibration then compares the frozen solver with the fixed fallback and determines whether to add it to the program registry. At deployment, label-free routing and local output checks either return a program answer or preserve the fallback. The following subsections describe these stages in order.

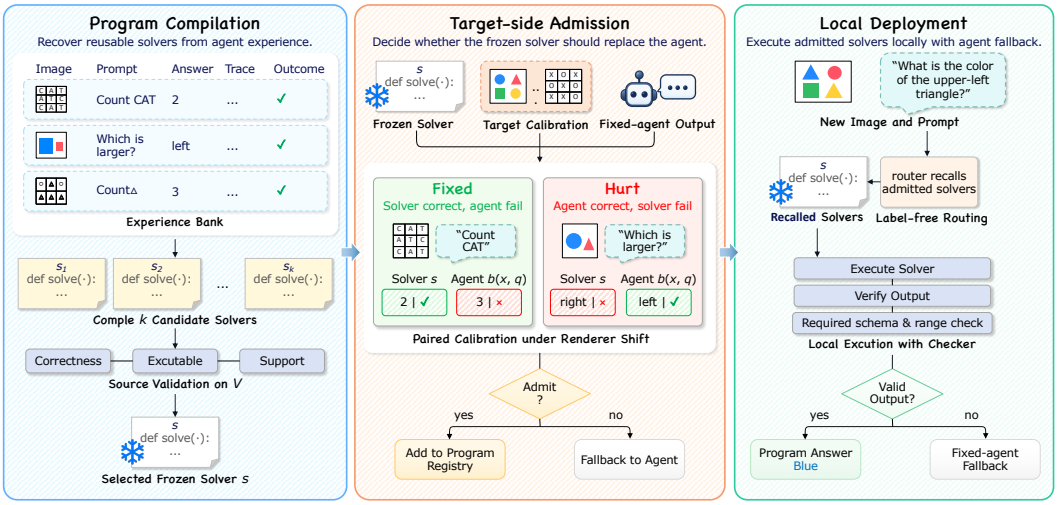


Figure 2: Trace2Prog's three stages. Compilation selects a frozen solver from candidates built from a bounded experience bank. Target-side calibration uses paired Fix/Hurt outcomes against the fixed tool agent to decide admission. At inference, label-free routing and local output checks either return a program answer or retain the fixed-agent fallback.

4.1 Compile: From Experience to Candidate Solvers

The compiler receives a fixed-size bank of prior agent executions for one registered contract. Each episode contains the image and prompt, the agent's answer and correctness, and a bounded excerpt of its execution trace, including code and tool feedback when available. Bounding both the number of episodes and the trace envelope makes candidate budgets comparable across source conditions and fixes the evidence available to each compilation attempt.

Successful traces can provide executable scaffolds; failed executions expose parsing and execution errors. Section 5 compares source-bank compositions under matched compilation budgets. Preprocessing packages the evidence without supplying a task-specific algorithm. Rather than retaining this evidence as request-time context, compilation distills it into a standalone procedure that can be frozen and replayed locally across requests. Appendix A.2 illustrates this transition for a rotation task.

Given the contract, source episodes, allowed Python libraries, and the required `solve(image, prompt)` interface, a language model independently emits k self-contained candidates. Independent draws expose multiple parsing and algorithmic hypotheses to the same downstream validation rule instead of committing deployment to one synthesis attempt. A candidate may inspect source images through sandboxed Python, but it cannot access source-validation, target calibration, or final rows. We remove surrounding generation text, check syntax and interface conformance, reject prohibited operations, and freeze the bytes of every executable candidate before evaluation. The default candidate budget is $k = 3$; Section 5 records the compiler-specific budgets used in each evaluation track.

Source validation tests cross-request usefulness. Every executable candidate is replayed in a network-disabled filesystem sandbox with a per-example timeout. The ledger records answers, abstentions, errors, baseline correctness, Fix, Hurt, and Net. A candidate is source-eligible if it provides at least five valid outputs, has selective precision at least 0.6, and achieves positive Net. Among eligible candidates, Trace2Prog maximizes Net, then minimizes Hurt, maximizes precision, and chooses the earliest replicate. This produces at most one source-selected solver per contract.

4.2 Calibrate: Target Replacement Utility

Target admission asks a different question: whether that solver should replace the designated fallback under the target renderer. Before calibration, we freeze the selected solver and checker and construct disjoint calibration V_t and final T_t splits for each contract–renderer pair. The source-selected solver

then runs on V_t. Calibration changes the admission decision, not the solver code. A source-selected program may be rejected because it introduces errors after a font, layout, color, or parameter shift.

For the frozen solver s, the admission rule is

$$A(s; V_t) = \mathbf{1}\{\text{precision}(s; V_t) \geq 0.8 \wedge \text{Net}(s; V_t) > 0 \wedge p_{\text{McN}}(s; V_t) \leq 0.1\}, \tag{2}$$

where p_{McN} is the one-sided exact McNemar test of whether Fix exceeds Hurt. In the controlled study, admission pools the 60 calibration examples across three renderer profiles. Precision constrains the quality of accepted answers, while the paired terms assess their value relative to the fallback. Appendix A.6 reports threshold sensitivity.

Admission applies to a fixed program, checker, fallback, and target setting. Changing any of these requires recalibration. Admitted units may execute on T_t; rejected units preserve the fallback. Appendix A records the corresponding artifact checks.

4.3 Deploy: Label-Free Routing and Local Execution

The registry pairs opaque program IDs with short cards describing observable interfaces—for example, "count a requested glyph; return one integer." Cards contain no benchmark label, gold answer, or request-specific metadata. A router VLM receives only the image and prompt and returns at most two IDs with one confidence score. In the refined policy, prompt rules route distinct interfaces directly. By contrast, ambiguous color interfaces use two stages: the VLM proposes a color contract, and a calibration-trained geometry classifier, referred to as the visual witness, distinguishes circle from rectangular-bar geometry. The witness uses image component, contour, projection, and appearance features. It can redirect the proposal to another admitted color solver; if its score or margin fails the gate, the policy abstains. Appendix A.9 gives the rules, program types, and representative decisions.

Recalled solvers execute in isolated local sandboxes. Trace2Prog deploys an answer only when route confidence passes the threshold and exactly one recalled solver returns an output satisfying its schema and range checks. Low confidence, zero or multiple valid outputs, abstention, timeout, execution error, or a malformed answer preserves the fixed-agent prediction. The checker detects interface violations and execution failures; semantic errors in a valid answer remain measurable through Hurt.

Contracts, interface cards, prompt rules, and checker schemas are specified components of the evaluation environment. The compiler generates solver code; the visual witness is fitted separately. Task-label-free deployment refers to online requests; offline selection and calibration use labels.

Solver execution requires no compiler-model call and can be reused across requests. End-to-end label-free deployment does use one lightweight VLM routing call per request; registered-contract dispatch omits that call. The router, visual witness, threshold, registry, checkers, and fallback are frozen before policy execution.

5 Experimental Setup

We organize the evaluation into three tracks. The first measures end-to-end label-free deployment on complete benchmark pools. The second isolates renderer shift under controlled contracts. The third tests whether target admission distinguishes useful programs from harmful ones. All tracks use paired Fix, Hurt, and Net relative to a fixed fallback.

5.1 Track I: Full-Benchmark Label-Free Deployment

The end-to-end evaluation covers all 1,215 TIR-Bench requests (Li et al., 2026) and 2,000 PuzzleVQA requests (Chia et al., 2024) with one shared Qwen3.6-Plus fallback. Twenty-four of 33 task families contain at least 12 fallback failures and enough requests for request-disjoint source, source-validation, target-calibration, and final roles; the remaining families are fixed passthroughs. Development and calibration rows also retain the fallback in the complete 3,215-request denominator.

Plus uses four candidates per eligible contract, including one source-side resample; the other compiler budgets are detailed in the appendix. Five Plus programs pass source selection, and two pass target admission. This track uses the observed benchmark renderer in Eq. (2).

Using this Plus configuration as one registry, we compare Plus, Kimi-K2.7-Code, GPT-5.6-Luna, and Claude Sonnet 5 with the same Qwen3.6-Flash router model. Every compiler uses temperature zero, a Top-2 limit, prompt checks, 388 calibration requests, and 1,299 shared final requests. For Luna and Claude, a shared geometry witness fitted on 24 supported and 84 unsupported calibration images disambiguates color contracts. Both registered-contract and label-free dispatch are scored on these same shared final requests; the other 1,528 requests retain the shared fallback. Router calibration requires positive routed Net and zero wrong-contract deployment, while solver Hurt is separately constrained by target admission. Plus and Kimi select threshold 0.9; Luna and Claude select 0.95. The admitted registries contain two, one, six, and three programs, respectively.

The geometry witness was developed after the initial router evaluation, fitted on calibration data, and frozen before retrospective replay on the already opened shared final requests.

5.2 Track II: Controlled Renderer Shift

We construct a controlled study from four benchmark-derived contracts selected during development for deterministic re-rendering, exact scoring, and matched source conditions: word-grid counting and localization from TIR-Bench, and shape-size and size-cycle reasoning from PuzzleVQA. The word-grid pair shares visual inputs but changes the answer schema; the PuzzleVQA pair shares shape and size primitives but changes the reasoning rule. Contract identities are frozen before generating or evaluating target roles. This track isolates renderer-sensitive deployment, while Track I measures full-benchmark scope.

We predeclare three profiles per contract: DejaVu, Roboto, and Lora for word grids; compact/cool, large/warm, and randomized parameters for PuzzleVQA. Each contract–profile pair contains 20 calibration and 40 final requests, yielding $4 \times 3 \times 20 = 240$ calibration and $4 \times 3 \times 40 = 480$ final requests. The fixed Qwen3.6-Plus tool agent supplies the fallback predictions. The evidence envelope and three-candidate budget are shared across source conditions. Final examples are unseen, but their profiles appear in target calibration, so this track measures calibrated rather than zero-shot renderer transfer.

We compare alternative inference and reuse mechanisms on shared requests. The controls include a frozen one-retry policy, direct three-shot prompting, per-query program generation, and success-only persistent compilation. The evidence-matched per-query control receives the same bounded historical episodes and source images as the compiler, but synthesizes anew for every final request. Retry, prompting, and per-query controls are retrospective on fixed final requests; success-only compilation follows the original frozen protocol. Appendix A details each control and its correspondence to FlowEvo and XSkill.

5.3 Track III: Admission Mechanism and Controls

Track III uses separately generated calibration and final requests for three mixed-source and three success-only attempts per contract; 18 of 24 are executable and ten are source-eligible. Each candidate is evaluated on 120 contract-specific final requests, yielding 2,160 candidate–request evaluations. The 0.8 precision and 0.1 McNemar thresholds predate both test blocks. Additional calibration-only threshold sweeps were defined after the first three contract results and before the size-cycle test; the primary gate remained unchanged. Appendix A.6 gives the chronology, threshold sweeps, and clustered intervals.

The FlowEvo-style success-only control holds the source pool, evidence cap, candidate budget, sandbox, scorer, and admission protocol fixed. Additional source-condition, XSkill-adaptation, and multi-image studies appear in the appendix.

6 Results and Analysis

6.1 RQ1: Can Persistent Programs Provide Selective End-to-End Improvement?

Table 1 compares four compiler registries under a shared routing backbone and evaluation protocol, with registry-specific calibrated thresholds. Under label-free routing, every compiler registry has positive Net with zero observed Hurt and zero wrong-contract deployment. Luna deploys on 150/1,299

shared final requests and raises benchmark-wide accuracy from 53.7% to 55.9%, a gain of 2.18 points. Relative to registered-contract Net on the same requests, label-free routing retains 100%, 100%, 97.2%, and 94.1% for Plus, Kimi, Luna, and Claude, or $125/129 = 96.9\%$ when Net is summed across compiler configurations.

Table 1: Benchmark-wide scoring with one Qwen3.6-Plus fallback (1,727/3,215, 53.7%). Each compiler has 1,299 shared final requests. The other 1,916 requests, including 388 router-calibration requests, retain the fallback. "Dep." counts deployed program answers. The registered-contract columns use direct dispatch. The refined label-free routing policy is evaluated retrospectively. Correct counts and percentages use the full 3,215-request denominator.

			Registered contract			Label-free		
Compiler	Admitted	Final N	Dep.	Correct (%)	F/H/N	Dep.	Correct (%)	F/H/N
Qwen3.6-Plus	2	1299	39	1742 (54.2)	15/0/+15	39	1742 (54.2)	15/0/+15
Kimi-K2.7-Code	1	1299	32	1735 (54.0)	8/0/+8	32	1735 (54.0)	8/0/+8
GPT-5.6-Luna	6	1299	157	1799 (56.0)	74/2/+72	150	**1797 (55.9)**	**70/0/**+70
Claude Sonnet 5	3	1299	71	1761 (54.8)	34/0/+34	67	1759 (54.7)	32/0/+32

Registered-contract dispatch measures the utility available from each registry when the contract is known. The refined label-free routing policy recovers $70/72$ of Luna's and $32/34$ of Claude's registered-contract Net while preserving zero observed Hurt. The 96.9% retention summarizes four separate compiler configurations. The router omits eleven otherwise valid outputs, all from ambiguous color interfaces; Appendix A.9 traces these decisions.

6.2 RQ2: How Does Persistent Reuse Compare with Alternatives?

Table 2 centers the four-way comparison under the fixed tool-agent fallback. The fallback answers 232 of 480 requests correctly. Regenerating a program for every request reaches 416, success-only persistent compilation reaches 418, and Trace2Prog reaches 447 without per-request program synthesis.

Table 2: Comparison on the same 480 controlled final requests. The primary panel uses the fixed tool-agent fallback. The lower panel is a retrospective sensitivity that keeps admission decisions calibrated against the fixed tool-agent fallback while substituting a one-retry fallback during replay. † denotes evaluation on opened request identities.

Method	Program use	Correct ↑	Net vs. fallback	Per-request synth.
Primary comparison: fixed tool-agent fallback				
Fixed tool-agent fallback	None	232	–	No
Evidence-matched per-query synthesis†	Regenerate	416	+184	Yes
Success-only compilation	Persistent	418	+186	No
Trace2Prog	**Calibrated persistent**	**447**	+215	**No**
Sensitivity: one-retry fallback†				
One-retry fallback	None	320	–	No
Success-only compilation	Persistent	433	+113	No
Trace2Prog	**Original admission†**	**455**	+135	**No**

Compared with evidence-matched per-query synthesis, Trace2Prog answers 31 more requests correctly, with 52 wins and 21 losses ($p = 3.71 \times 10^{-4}$). The gain is 6.46 percentage points, with a clustered 95% interval of $[3.13, 10.00]$. The per-query control uses 1,948 model calls, whereas Trace2Prog reuses programs after offline compilation and calibration. Against success-only compilation, Trace2Prog gains 29 correct answers (45 wins, 16 losses; $p = 2.64 \times 10^{-4}$). This comparison with success-only compilation pertains to the selected mixed bank as a whole. Direct three-shot prompting reaches 234/480, compared with 173/480 for direct zero-shot prompting and 232/480 for the tool-agent fallback. Appendix A.4 reports the prompting controls.

With the stronger one-retry fallback, the agent reaches 320/480. Replaying the originally admitted Trace2Prog programs reaches 455, with Net +135 (145 Fix, 10 Hurt).

On a separate 516-request holdout with matched fallback, frozen Plus programs and a cached-experience XSkill adaptation respectively score 59 and 55 on TIR ($p = 0.597$), and 141 and 190 on PuzzleVQA ($p = 1.24 \times 10^{-7}$). Frozen programs cover 20/256 TIR and 20/260 PuzzleVQA requests. XSkill performs retrieval and adaptation across the holdout using 3,839 chat calls and 1,100 embedding requests. This comparison exposes the trade-off between selective local execution and broader online adaptation. Appendix A.3 gives the protocol.

6.3 RQ3: Does Target Calibration Distinguish Useful Candidates?

Table 3 compares admission rules across 18 executable candidates, each replayed independently on its 120 relevant requests from the separate Track III final set. Because candidates for the same contract overlap, totals aggregate 2,160 candidate–request evaluations rather than one 480-request policy. With no gate, candidates produce 302 Hurt. Source eligibility reduces this to 87, and applying the target gate after source eligibility retains seven candidates with 388 Fix and 24 Hurt. The final row applies source eligibility followed by target admission independently to each candidate; it does not evaluate the one-solver-per-contract selection policy used in deployment.

Table 3: Comparison of admission rules on 18 executable programs. Totals sum independent outcomes over 2,160 candidate–request evaluations. They are not end-to-end policy accuracy on 480 unique requests. Target criteria use a separate 240-request calibration role.

Candidate filter	Retain	Outputs	Fix/Hurt	Aggregate Net
None	18/18	1523	479/302	$+177$
Source eligibility	10/18	1044	431/87	$+344$
Target precision (≥ 0.8)	9/18	902	437/31	$+406$
Target paired utility ($\mathrm{McN} \leq 0.1$)	8/18	782	406/24	$+382$
Target precision + paired	8/18	782	406/24	$+382$
Source eligibility + target gate	**7/18**	**751**	**388/24**	$\mathbf{+364}$

These totals aggregate overlapping candidate outcomes; Table 2 reports end-to-end deployment. Precision-only yields $+406$ Net with 31 harms; the sequential filter retains source validation with $+364$ Net and 24 harms. It therefore trades Net for fewer harms rather than maximizing Net. Deployment selects at most one solver per contract. Appendix A.6 reports profile checks, operating points, and clustered intervals.

7 Limitations

The controlled study covers four benchmark-derived contracts and calibrated renderer profiles, not zero-shot transfer to arbitrary visual domains. Target admission requires labeled examples and provides empirical operating points, not formal risk guarantees. Compiler yield varies across models, so program coverage can remain the dominant source of error.

Trace2Prog assumes reusable behavior can be expressed through a bounded program interface and checked outputs. Larger registries require stronger contract discovery and routing; long-horizon or stateful tasks may not fit the same local execution model. These settings remain directions for broader experience reuse.

8 Conclusion

Trace2Prog separates program construction, target replacement, and request-level application. On 480 controlled final requests, it raises correctness from 232 to 447 without per-query synthesis. Across 2,160 candidate–request evaluations, sequential source eligibility and target admission reduce aggregate Hurt from 87 to 24 and raise aggregate Net from $+344$ to $+364$. A separate four-compiler analysis finds selective gains of 8–70 correct answers.

Conditional reuse, rather than compilation alone, is the central result: separating source evidence, target utility, and request applicability exposes replacement risk and preserves the agent under uncertainty.

AI USE STATEMENT

Generative AI tools assisted with manuscript organization and language polishing, literature retrieval and discovery, code development, figure prototyping, and feedback during research execution. The authors designed the method and experiments, ran and inspected all evaluations, and verified the resulting text, code, citations, figures, and reported results. The authors take responsibility for the final content of the paper.

REPRODUCIBILITY STATEMENT

Appendix A documents the data roles, protocols, execution budgets, and additional results. Our anonymous repository provides the implementation, frozen configurations and programs, deterministic renderers, tests, and a verifier that recomputes, rerunning model calls requires official data and compatible endpoints.

REFERENCES

Homayoun Afshari, Pietro Basci, Alessandro Russo, and Lia Morra. Think-verify-revise: Neuro-symbolic visual reasoning with vision-language models and dynamic logic tensor networks. *arXiv preprint arXiv:2609.05388*, 2026. URL https://arxiv.org/abs/2609.05388.

Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. Large language models as tool makers. In *International Conference on Learning Representations*, 2024.

Yew Ken Chia, Vernon Toh, Deepanway Ghosal, Lidong Bing, and Soujanya Poria. PuzzleVQA: Diagnosing multimodal reasoning challenges of language models with abstract visual patterns. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 16259–16273, 2024.

Yonatan Geifman and Ran El-Yaniv. SelectiveNet: A deep neural network with an integrated reject option. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

Tanmay Gupta and Aniruddha Kembhavi. Visual Programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

Guanyu Jiang, Zhaochen Su, Xiaoye Qu, and Yi R. Fung. XSkill: Continual learning from experience and skills in multimodal agents. In *Proceedings of the International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2603.12056.

Ming Li, Jike Zhong, Shitian Zhao, Haoquan Zhang, Shaoheng Lin, Yuxiang Lai, Chen Wei, Konstantinos Psounis, and Kaipeng Zhang. TIR-Bench: A comprehensive benchmark for agentic thinking-with-images reasoning. In Paolo Favaro, Zuzana Kukelova, Atsuto Maki, Anna Rohrbach, Konrad Schindler, and Federico Tombari (eds.), *Computer Vision - ECCV 2026 - 19th European Conference, Malmö, Sweden, September 8-12, 2026, Proceedings, Part XVI*, volume 17016 of *Lecture Notes in Computer Science*, pp. 464–481. Springer, 2026. URL https://doi.org/10.1007/978-3-032-37383-0_27.

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. Chameleon: Plug-and-play compositional reasoning with large language models. In *Advances in Neural Information Processing Systems*, 2023.

Zijian Lu, Yiping Zuo, Yupeng Nie, Xin He, Weibei Fan, Lianyong Qi, and Shi Jin. ContractSkill: Repairable contract-based skills for multimodal web agents. *arXiv preprint arXiv:2603.20340*, 2026. URL https://arxiv.org/abs/2603.20340.

Zeyu Ren, Ling Yue, Ran Li, Yishu Wang, Shengxiang Xu, Hanmo Liu, Shaowu Pan, and Shimin Di. FlowEvo: Self-evolving agents through the co-evolution of workflows and executable skills. In *Conference on Language Modeling*, 2026. URL https://arxiv.org/abs/2607.21596.

Qi Song, Honglin Li, Yingchen Yu, Haoyi Zhou, Lin Yang, Song Bai, Qi She, Zilong Huang, and Yunqing Zhao. CodeDance: A dynamic tool-integrated MLLM for executable visual reasoning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 19186–19195, 2026.

Dídac Surís, Sachit Menon, and Carl Vondrick. ViperGPT: Visual inference via python execution for reasoning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023.

Liyan Tang, Cyrus Rashtchian, Chun-Sung Ferng, Andrew Tomkins, Da-Cheng Juan, and Tu Vu. WikiSkill: Compiling agent experience into persistent knowledge for skill evolution. *arXiv preprint arXiv:2608.27454*, 2026. URL https://arxiv.org/abs/2608.27454.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Zhiruo Wang, Daniel Fried, and Graham Neubig. TroVE: Inducing verifiable and efficient toolboxes for solving programmatic tasks. *arXiv preprint arXiv:2401.12869*, 2024. URL https://arxiv.org/abs/2401.12869.

Shouang Wei, Houcheng Min, Xinpeng Dong, Xin Lin, Sen Cui, Bo Jiang, Zhongxiang Dai, Kun Kuang, Guandong Xu, Fei Wu, and Min Zhang. MetaForge: A self-evolving multimodal agent that retrieves, adapts, and forges tools on demand. *arXiv preprint arXiv:2606.01801*, 2026a. URL https://arxiv.org/abs/2606.01801.

Yangbo Wei, Zhen Huang, Shaoqiang Lu, Junhong Qian, Qifan Wang, Chen Wu, and Lei He. SkillSmith: Co-evolving skills and tools for self-improving agent systems. *arXiv preprint arXiv:2606.01314*, 2026b. URL https://arxiv.org/abs/2606.01314.

Feng Xiong, Zengbin Wang, Yong Wang, Xuecai Hu, Jinghan He, Liang Lin, Yuan Liu, and Xiangxiang Chu. Ace-Skill: Bootstrapping multimodal agents with prioritized and clustered evolution. *arXiv preprint arXiv:2605.08887*, 2026. URL https://arxiv.org/abs/2605.08887.

Duling Xu, Zheng Chen, Zaifeng Pan, Jiawei Guan, Dong Dong, Jialin Li, and Bangzheng Pu. SkillSmith: Compiling agent skills into boundary-guided runtime interfaces. *arXiv preprint arXiv:2605.15215*, 2026. URL https://arxiv.org/abs/2605.15215.

Zhengyuan Yang, Linjie Li, Jianfeng Wang, Kevin Lin, Ehsan Azarnasab, Faisal Ahmed, Zicheng Liu, Ce Liu, Michael Zeng, and Lijuan Wang. MM-REACT: Prompting ChatGPT for multimodal reasoning and action. *arXiv preprint arXiv:2303.11381*, 2023.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

Lifan Yuan, Yangyi Chen, Xingyao Wang, Yi Fung, Hao Peng, and Heng Ji. CRAFT: customizing llms by creating and retrieving from specialized toolsets. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=G0vdDSt9XM.

Kangning Zhang, Shuai Shao, Qingyao Li, Jianghao Lin, Lingyue Fu, Shijian Wang, Wenxiang Jiao, Yuan Lu, Weiwen Liu, Weinan Zhang, and Yong Yu. MMSkills: Towards multimodal skills for general visual agents. *arXiv preprint arXiv:2605.13527*, 2026. URL https://arxiv.org/abs/2605.13527.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

A Supplementary Experiments and Protocol Details

A.1 Controlled-contract selection and temporal order

The four Track II contracts are selected using development artifacts because they support deterministic re-rendering, exact answer generation, and matched mixed and success-only compilation. The selection therefore targets mechanism identification rather than benchmark representativeness. After contract choice, we generate and hash-freeze all 720 target images, partition each contract–profile block into calibration and final roles, and freeze the programs, checker, fallback, and admission rule before final execution. No outcome from these target final roles participates in contract selection. Track I separately evaluates end-to-end scope over the complete benchmark pool.

Track I roles are disjoint by image–question identity. Because the released benchmarks can attach different questions to the same image, calibration and final roles share 38 image hashes across 68 final requests; no exact image–prompt pair overlaps. The controlled Track II study instead uses 720 unique newly rendered images.

All 720 target images have unique hashes and overlap none of the 3,159 historical images checked during construction. Each deployment record binds the program hash, contract, renderer profiles, fallback configuration, checker, and admission rule. Source experience is used for construction, held-out source examples select among frozen candidates, and target calibration evaluates the selected program without changing its bytes.

A.2 From execution experience to a reusable program

Figure 3 gives a concrete rotation example. The original agent trajectory repeatedly estimates image geometry yet returns the wrong option. Compilation converts the bounded execution evidence into a standalone solver that extracts contours, constructs rotation hypotheses, and matches them against the answer choices. The resulting procedure is no longer tied to that interaction: after freezing, it can execute locally on new requests through the same `solve(image, prompt)` interface. Target calibration remains a separate step that determines whether this reusable procedure should replace the designated fallback.

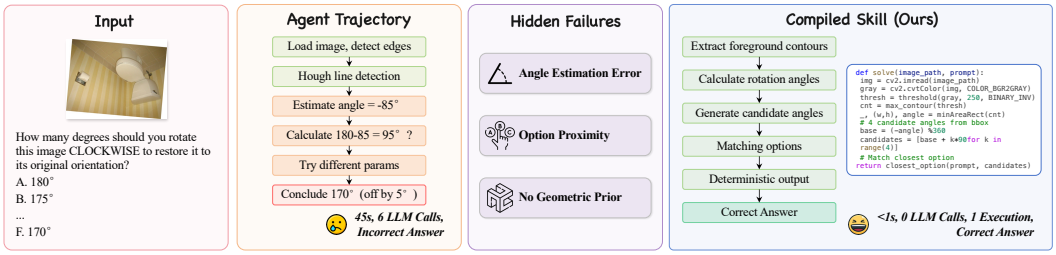


Figure 3: Illustrative transition from an agent execution to a frozen rotation solver. The trajectory contains task-specific visual operations but reaches an incorrect answer; the compiled program turns those operations into a standalone geometric procedure that can be executed repeatedly without calling the solver model. Timing and call-count annotations describe this example, not aggregate measurements. Admission is evaluated separately on held-out target calibration data. The displayed code is schematic pseudocode.

A.3 External comparison protocol and failure accounting

We reuse the official XSkill image-captioning, trajectory-summary, experience-critique, similarity-merge, skill-generation and refinement components. At inference, the adaptation retains visual task decomposition, top-three retrieval per subtask, experience rewriting and skill adaptation. One global memory library is built per benchmark from 180 TIR-Bench and 208 PuzzleVQA source episodes. These are the same cached agent episodes available to the compiler, with a 900-character execution-evidence envelope and a 300-character agent-answer envelope. One cached rollout per

Table 4: Correct answers on the sampled common holdout. All methods use Qwen3.6-Plus. † denotes retrospective reuse of the same fresh-agent predictions: for XSkill on empty answers, and for frozen programs when not applied.

Policy	TIR / 256	PuzzleVQA / 260
Fresh tool agent	49	133
XSkill adaptation	34	171
XSkill + matched fallback†	55	190
Frozen Plus programs + matched fallback†	59	141

source replaces native online multi-rollout collection; intermediate tool images absent from the cache are not reconstructed. We refer to this implementation as a cached-experience XSkill adaptation.

Source memory is frozen before held-out evaluation. A fixed hash ordering of case identities selects up to 20 requests in each of 26 eligible task families, including those with no retained solver. Two TIR families have only 18 held-out rows, giving 256 TIR and 260 PuzzleVQA requests. Source and target image–question pairs are disjoint; repeated images with different questions can occur. All questions and failures remain in the denominator.

Both arms use Qwen3.6-Plus without thinking, temperature 0.2, at most eight API turns and eight Python executions per query, and a 180-second execution deadline. Individual executor responses are capped at 4,096 tokens. Memory construction allows up to 12,288 output tokens; online retrieval and adaptation use the corresponding official stage caps, up to 8,192 tokens. The original embedding service is replaced by 1,024-dimensional text-embedding-v4. Python subprocesses are stateless, with per-query files persisting in an isolated, network-disabled workspace. API previews are limited to two million pixels; images smaller than 28 pixels along either dimension receive white padding. Original pixels remain accessible to code. XSkill's context preparation precedes the execution deadline, so equal executor limits do not imply equal total computation.

The fresh agent leaves 167 TIR and 68 PuzzleVQA requests unanswered; XSkill leaves 165 and 42 unanswered. These include turn-limit and timeout outcomes and are scored incorrect in the raw comparison. The retrospective XSkill fallback retains the same fresh-agent prediction only for empty answers. The matched-fallback program policy retains that prediction whenever the two-program Plus registry is not applied; its admission decision is unchanged. Solver coverage is 20/256 on TIR (rotation) and 20/260 on PuzzleVQA (shape size). With matched fallback, the solver policy wins 18 and loses 14 TIR pairs against XSkill (two-sided exact McNemar $p = 0.597$), versus 19 wins and 68 losses on PuzzleVQA ($p = 1.24 \times 10^{-7}$). The comparison uses one sampled split; the TIR difference is not statistically significant.

Completed query-phase chat calls total 2,919 for the fresh agent and 3,839 for XSkill. XSkill additionally uses 1,100 completed embedding requests. These figures include its retrieval and adaptation chats but exclude offline memory construction, failed-request charges without returned usage, and acquisition of a separate fallback answer. Table 4 reports paired accuracy under these recorded online budgets.

A.4 Direct visual prompting controls

On the 480 Track II final requests, direct zero-shot and three-shot prompting reach 173 and 234 correct answers, respectively, compared with 232 for the fixed tool-agent fallback and 447 for Trace2Prog. Both controls use Qwen3.6-Plus without tools; three-shot additionally receives labeled calibration examples.

A.5 RENDERER-SHIFT RESULTS BY CONTRACT

Table 5: Three-profile renderer-shift evaluation. Each contract has 120 final requests. Panel A is a retrospective sensitivity using one gold-free retry on observable failures; Panel B uses the original fixed tool-agent fallback.

Panel	Contract	Agent	Success-only control	Trace2Prog	Fix/Hurt/Net
A: retry	Word count	62	120	116	55/1/+54
	Word locate	73	109	112	40/1/+39
	Shape size	107	107	120	13/0/+13
	Size cycle	78	97	107	37/8/+29
	Total	**320**	**433**	**455**	**145/10/+135**
B: fixed	Word count	37	120	116	80/1/+79
	Word locate	39	101	107	69/1/+68
	Shape size	104	104 (reject)	120	16/0/+16
	Size cycle	52	93	104	58/6/+52
	Total	**232**	**418**	**447**	**223/8/+215**

A.6 GATE MECHANISM AND THRESHOLD SENSITIVITY

We compare precision, paired utility, their conjunction, and the original conservative gate on the same 18 executable programs and 240 calibration requests. Precision thresholds are swept over $\{0, 0.5, 0.6, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 1\}$ and paired-test cutoffs over $\{0.01, 0.025, 0.05, 0.1, 0.2, 0.5, 1\}$. The primary thresholds predate both test blocks; the unified sensitivity grid was assembled after the original three-contract outcomes and before the size-cycle test. Every operating point is nevertheless computed from calibration only. Table 6 reports two calibration-defined selection rules. "Risk" maximizes calibration Net subject to $\mathrm{Hurt}/N \leq 1\%$; "coverage" selects the nearest point to 40% calibration coverage. Intervals resample unique questions within contract and renderer while retaining all candidate outcomes for each question. The original conservative gate was frozen before the four-contract analysis and adds minimum-support, Wilson lower-bound, and per-profile checks.

Table 6: Calibration-selected operating points and held-out outcomes. The original conservative gate predates the four-contract analysis and adds support, Wilson, and per-profile checks. Net intervals are question-clustered 95% bootstrap intervals. Coverage uses all 1,080 calibration and 2,160 final candidate–request evaluations as denominators.

Selector	Policy	Cal. cov.	Cal. Hurt	Test cov.	Test Hurt	Test Net [95%]
Risk $\leq 1\%$	Precision	29.1	10	28.3	21	275 [243,308]
	Paired utility	0.0	0	0.0	0	0 [0,0]
	Precision + paired	28.6	9	27.9	17	298 [268,328]
	Full, no profile	26.8	8	26.5	17	280 [252,306]
	Original conservative	30.5	9	29.6	21	317 [282,350]
Coverage $\approx 40\%$	Precision	43.0	21	41.8	31	406 [363,449]
	Paired utility	37.4	15	36.2	24	382 [342,421]
	Precision + paired	37.4	15	36.2	24	382 [342,421]
	Full, no profile	41.1	20	40.3	31	388 [347,430]
	Original conservative	36.0	15	35.2	28	341 [303,378]

The precision sweep reaches low-risk, low-coverage points; paired utility adds intermediate points with substantially higher Net. Their conjunction contains the useful points from both component sweeps. Absolute support and Wilson checks remove Net without reducing Hurt at the frozen thresholds. Profile checks do change the four-contract decisions: relative to the no-profile conservative gate, they reduce Hurt from 24 to 21 while reducing Net from $+364$ to $+317$.

A.7 CANDIDATE-LEVEL GATE OUTCOMES

Table 7 lists all 18 executable programs in the mechanism study. Six additional frozen attempts fail static execution and are excluded from program-level discrimination. Each identity-disjoint

test block opens only after its candidate bytes, roles, and primary thresholds are frozen; the unified sensitivity selectors never modify a candidate. These candidate-level blocks are separately generated identities: for example, their word-count fallback answers 36/120 requests correctly, whereas the renderer-transfer panel in Table 5 uses a 37/120 word-count fallback. Candidate Nets therefore should not be combined with the per-contract rows of that panel.

Table 7: Program-level calibration and held-out outcomes. "Source-eligible" marks source eligibility; "admit" uses precision plus paired utility before source conditioning.

Bank	Contract	Rep.	Source-eligible	Cal. Net	Final Net / Admit
Mixed	Shape size	1	yes	-2	$+2$ / no
Mixed	Shape size	2	yes	$+7$	$+10$ / yes
Mixed	Size cycle	1	yes	$+21$	$+47$ / yes
Mixed	Size cycle	2	no	-21	-38 / no
Mixed	Size cycle	3	no	-30	-57 / no
Mixed	Word count	1	yes	$+43$	$+84$ / yes
Mixed	Word count	2	yes	$+18$	$+39$ / yes
Mixed	Word locate	2	yes	-17	-46 / no
Mixed	Word locate	3	yes	$+43$	$+74$ / yes
Success	Shape size	1	no	-49	-89 / no
Success	Shape size	3	yes	$+5$	$+24$ / no
Success	Size cycle	1	yes	$+19$	$+45$ / yes
Success	Size cycle	2	no	0	0 / no
Success	Size cycle	3	no	$+1$	-1 / no
Success	Word count	1	yes	$+37$	$+65$ / yes
Success	Word count	3	no	$+12$	$+18$ / yes
Success	Word locate	1	no	0	0 / no
Success	Word locate	2	no	0	0 / no

A.8 Compiler scope before target admission

Table 8 compares four frozen source-selected compiler registries against the same Qwen3.6-Plus fallback and scorer on shared full-benchmark final requests. Source roles and candidate budgets follow each compiler's frozen run. This table isolates compilation scope before target admission.

Table 8: Source-selected compiler comparison with one shared agent baseline; no target-side admission. All 33 task families enter the denominator.

Compiler	Benchmark	N	Exec. N	Baseline	Agent + program	Fix	Hurt	Net
Qwen3.6-Plus	TIR-Bench	1215	87	222	247	28	3	+25
Qwen3.6-Plus	PuzzleVQA	2000	112	1505	1528	33	10	+23
Kimi-K2.7-Code	TIR-Bench	1215	0	222	222	0	0	0
Kimi-K2.7-Code	PuzzleVQA	2000	112	1505	1524	19	0	+19
GPT-5.6-Luna	TIR-Bench	1215	226	222	316	104	10	+94
GPT-5.6-Luna	PuzzleVQA	2000	560	1505	1677	185	13	+172
Claude Sonnet 5	TIR-Bench	1215	31	222	242	21	1	+20
Claude Sonnet 5	PuzzleVQA	2000	168	1505	1560	60	5	+55

In a separate compiler-specific PuzzleVQA role before target admission, the selected Plus shape-size solver obtains Fix/Hurt/Net $= 13/19/-6$ on 68 requests; all other PuzzleVQA families retain the fallback. This role is distinct from the shared full-benchmark evaluation in Table 8.

Selected-program scope is 199/3,215 requests for Plus and Claude, 112 for Kimi, and 786 for Luna. All four source-only policies have positive aggregate Net, but Plus, Luna, and Claude introduce 13, 23, and 6 harmful replacements, respectively, motivating target admission.

Compiler-specific confirmation. In a separate target-admitted confirmation, each registry is evaluated against the Qwen3.6-Plus fallback on its compiler-specific frozen final role. Plus admits rotation-game and shape-size, with Nets $+26$ and $+15$. Kimi admits shape-size and obtains $15/0/+15$; its selected size-grid program is rejected by paired calibration ($p = 0.25$). Luna uses a bounded direct

adapter because its endpoint does not forward code-interpreter calls; it selects 14 contracts, admits six, and obtains $144/7/+137$. Claude selects four contracts, admits three, and obtains $73/1/+72$.

Shared final requests. Table 1 places all four admitted registries on the same 1,299 shared final requests. Plus, Kimi, Luna, and Claude obtain label-free Nets $+15, +8, +70, +32$, with zero Hurt and zero wrong-contract deployment. The Luna and Claude label-free results use the retrospective color-geometry witness described in Section 5.

Separate Plus confirmation. A two-program Plus registry was independently frozen and evaluated on a larger set of 2,083 routed final requests. It deploys 61 outputs and obtains Fix/Hurt/Net $= 36/0/+36$.

For the retrospective Luna and Claude label-free results in Table 1, the color-geometry witness uses deterministic connected-component, contour, projection, and appearance statistics. A counterfactual logistic model is fitted on 12 circle-color positives, 12 rectangular-bar-color positives, and 84 unsupported color-geometry calibration images. Qwen3.6-Flash first proposes an ambiguous color contract; the witness accepts its highest-scoring geometry only above a frozen score of 0.85 and margin of 0.5, otherwise abstaining.

On a second identity-disjoint split with the same baseline, candidate budget, and scorer, Plus obtains Fix/Hurt/Net $= 29/1/+28$ on rotation and $15/5/+10$ on shape size; Claude obtains $39/0/+39$ and $15/2/+13$. Kimi produces no executable candidate on this split.

A.9 Admitted program types and routing omissions

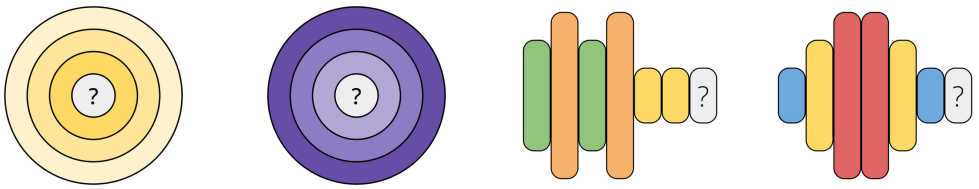
The Plus, Kimi, Luna, and Claude registries admit two, one, six, and three programs, respectively. Across them, the executable interfaces cover upright restoration, polygon-size relations, circle color–size relations, rectangular-bar height–color and height–number relations, and cyclic size patterns. Every entry uses the same frozen local `solve(image, prompt)` interface with a contract-specific checker and fallback; exact program IDs and source are provided in the anonymous repository.

The routing funnel in Table 9 separates contract coverage, solver answerability, and routing. Luna's ten solver abstentions also occur under registered-contract dispatch; the router accounts only for seven omissions among otherwise answerable requests. Claude has four router omissions and no solver abstention. All eleven omissions occur in ambiguous color interfaces, and none produces a wrong-contract deployment in this replay.

Table 9: Final routing funnel on the 1,299 common identities. "Contract-covered" counts requests mapped to an admitted program; "solver-valid" uses direct registered-contract execution.

Compiler	Contract-covered	Solver-valid	Router omissions	LF deployed
Qwen3.6-Plus	39	39	0	39
Kimi-K2.7-Code	32	32	0	32
GPT-5.6-Luna	167	157	7	150
Claude Sonnet 5	71	71	4	67

Figure 4 shows matched examples. The accepted circle and bar cases score 0.931 and 0.867 for their respective geometry, above the frozen 0.85 gate. Three monochromatic circle instances share a score of 0.324, while four symmetric bar instances score 0.718–0.807. Their prompts are identical at the interface level ("missing color"), so the geometry witness conservatively abstains and preserves the fallback. This selective behavior converts uncertain visual matching into lower coverage rather than an incorrect persistent override.



Circle: route (0.931) Circle: abstain (0.324) Bars: route (0.867) Bars: abstain (0.807)

Figure 4: Examples of selective routing between two color-pattern programs. The routing policy invokes the corresponding frozen program only when its geometry score and margin pass the calibrated requirements; otherwise, it abstains and retains the fixed tool-agent fallback.

A.10 Admission precision and uncertainty

Pooled final precision and Wilson 95% lower bounds for the admitted Trace2Prog programs are respectively 0.967/0.917 (word count), 0.990/0.947 (word locate), 1.000/0.969 (shape size), and 0.894/0.824 (size cycle). These describe observed selective reliability after admission, not future-shift guarantees.

A.11 Evidence-matched per-query synthesis

The per-query generator receives the exact bounded mixed historical experience used by the persistent compiler, including source images, labels, and 1,800-character execution excerpts. On the same 480 final requests, this evidence-matched control obtains 407 correct raw answers and 416 with the fixed fallback (Fix/Hurt/Net $= 196/12/+184$). Persistent Trace2Prog obtains 447 (223/8/+215), winning 52 and losing 21 pairs ($p = 3.71 \times 10^{-4}$). The evidence-matched control uses 1,948 completed model calls and 1,245 code executions. This control was frozen after the final requests had been opened.

A.12 Additional label-free routing studies

Before the shared four-compiler evaluation in Table 1, an independently frozen Plus confirmation used a two-program registry. We also evaluated whether a separate four-program registry could be selected without task metadata. A counterfactually trained local router produces no unsupported deployment but retains only 46.8% of oracle Net. We therefore freeze a different policy before new-role generation: a frozen router VLM recalls at most two programs from contract cards, every candidate executes, and only a unique schema-valid answer supported by a deterministic relation witness may replace the fallback. The 50-row smoke test retains all 31 oracle-correct deployments with no false deployment.

Contract cards expose only the observable interface. For example, the count card reads "Count a requested letter or digit in a dense repeated-glyph grid; return one integer." Cards contain no benchmark name, gold answer, or request-specific metadata; the router returns opaque program IDs.

Table 10 reports the prospective confirmation across three independently generated splits.

Table 10: Prospective registered-contract versus label-free confirmation. Each split contains 40 registered and 40 unsupported scored requests plus eight malformed requests. "Label Net" uses the correct registered contract.

Split	Agent	Label-free	LF Fix/Hurt/Net	Label Net	Retention
801	49	56	7/0/+7	+7	100%
809	52	60	8/0/+8	+8	100%
821	43	57	14/0/+14	+15	93.3%
Total	144	173	29/0/+29	+30	96.7%