COMPOSITION DISCOVERY IN TOOL WORLDS BY ACTIVE EXPLORATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Agents usually discover connections between tools, which we call handoffs, while solving a task, leaving the knowledge tied to specific trajectories. We investigate whether an agent can instead explore a tool world, a set of callable functions together with the data behind them, before any task, with the explicit goal of building reusable knowledge about its tools. Task-free exploration faces two bottlenecks: the combinatorial number of candidate handoffs, and the volatility of tool data. Because experiments require valid inputs from earlier calls, every change of the data invalidates the inputs collected so far. We introduce ICHNAEA, an explorer that maintains a Beta belief over candidate handoffs and selects the calls whose outcomes are expected to be most informative. To overcome data volatility, ICHNAEA stores each successful call as a recipe and, when the data change, re-executes its recipes to rebuild fresh inputs. We evaluate ICHNAEA on procedural tool worlds, where the true handoffs are known and names can be hidden, and on AppWorld, a benchmark of simulated everyday apps for language-model agents. In procedural worlds, ICHNAEA learns handoffs that complete every target workflow in unseen data, with no false handoff, while without rebuilding its inputs the explorer completes 0–73% depending on how much the names reveal. A language-model agent given these handoffs reaches every target, against 0–89% without them. On AppWorld, documentation alone already lets the agent reach nearly every goal, and some calls succeed only by coincidence. This points to two limits outside generated worlds: what exploration adds depends on how much of a suite's composition its documentation leaves unstated, and without known true handoffs, discovery is hard to measure.

1 INTRODUCTION

Software agents act by calling functions, and a task often needs the output of one function as the input of another. In AppWorld (Trivedi et al., 2024), a benchmark of nine simulated everyday applications used to evaluate language-model agents, liking a song takes several such steps. The agent first logs in. The user's profile returns an email, the login function accepts that email as a username, and a password lookup supplies the password. The login returns an access token, the token opens the song library, and a song ID from the library can be passed to the function that likes a song (Figure 1). Each step passes a value produced by one function to another. We call such a connection a *handoff* and write it (p, f, c, q): the value in field f of producer p's response is passed as parameter q of consumer c. A *tool world* is a set of callable functions together with the data behind them.

Agents usually find handoffs while solving a task, so what they learn is tied to that task. Documentation helps, but it describes functions one at a time. In AppWorld, a shared name, a description or a shared word implies 88% of the handoffs that human solutions use. The remaining 12% include every handoff that supplies a login username, and 98% of tasks need one of these. We ask whether an agent can find handoffs before any task, by calling its tools, and keep them as knowledge that later tasks can reuse.

Two obstacles make this difficult. The first is the number of candidates. The 355 functions in our AppWorld scope have 1,515 output fields and 1,276 input parameters, about 1.93 million field-to-parameter pairs, and one call may bind several parameters. The second obstacle is that experiments

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.

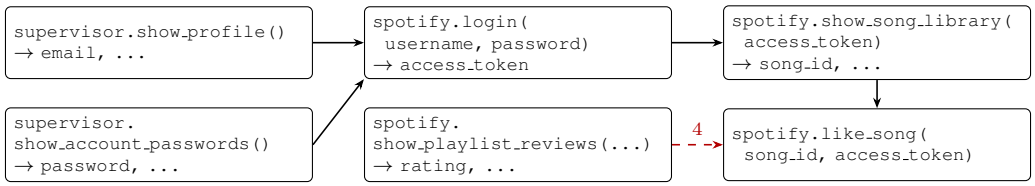


Figure 1: Selected handoffs and a coincidence in AppWorld. The profile's email supplies the login username; a login token and a library song ID support later calls. A review rating of 4 can also be accepted as a song ID, simply because song 4 exists. The login description, "Your account email", offers a clue even though it does not list the producing function.

need inputs. To test whether a song ID can be passed to the function that likes songs, the explorer must hold a valid access token and a song ID from the current data. Tool worlds change: records are created and deleted, and sessions expire. Values collected earlier then stop working, and knowing that a handoff exists no longer provides the inputs needed to test what it enables. In procedural worlds whose data change every 500 calls, on average fewer than 2% of the argument-bearing calls made by our first explorer had all their inputs available.

We present ICHNAEA, named after the Greek goddess of tracking, because it follows values from the response fields in which they are produced to the parameters of other functions that may accept them. For example, if a profile function returns the value `alice@example.com` in an `email` field and a login function may accept the same value as its `username` argument, ICHNAEA treats `profile.email` → `login.username` as a candidate handoff. It maintains a Beta posterior for each such candidate, representing its current confidence that the handoff is valid. When a call succeeds, the handoffs used to construct that call receive positive evidence; when it fails, ICHNAEA uses the service's error message to identify which argument was rejected and assigns negative evidence to the corresponding handoff rather than penalizing all handoffs in the call.

Before a candidate handoff has been tested directly, ICHNAEA ranks it using the values observed during exploration. In particular, if values produced by a source field repeatedly occur among the values that responses return under the target parameter's name, the handoff becomes a strong candidate. This is analogous to inclusion-dependency tests used in databases to discover foreign-key relationships (Bauckmann et al., 2007): the values of a foreign key are contained in those of the key it references. The test relies on names that match. Where they do not, ICHNAEA learns the correspondence from executed calls: once a value from a field with one name is accepted by a parameter with another, it proposes that pairing of names at every other function that uses them. Because a tool suite may contain millions of possible source-field–target-parameter pairs, ICHNAEA cannot test them all. Instead, it chooses calls according to the expected information gained from their outcomes, prioritizing experiments that are most useful for reducing uncertainty about the handoff graph. Documentation or other prior knowledge may influence how likely ICHNAEA expects a call to succeed, but it does not determine how informative the observed outcome is. This separation allows execution evidence to correct even a strong but incorrect prior.

Choosing informative calls does not help if the calls cannot be made. ICHNAEA therefore separates what it knows about the tools from the values it currently holds. Held values are discarded when the data change. Each successful call is also stored as a *recipe*: the function called and, for each argument, the function and field that supplied it. After a change, ICHNAEA calls every function that takes no input, which we call a *root*, and then re-executes its recipes in dependency order, so that the outputs of earlier recipes supply the inputs of later ones. Re-execution provides fresh inputs for new experiments and tests the recipes' handoffs again on new data.

Measuring discovery requires knowing what there is to discover. We therefore generate tool worlds whose handoff graph the evaluator knows and the explorer does not (Section 5.1). We vary what names reveal. With real names, every handoff is an exact name match. In the *masked* condition, the parameter of one necessary transition per chain is renamed, so names alone complete no chain. With *independent* names, every field and parameter is renamed. We count the handoffs witnessed by a successful call, the handoffs reproduced in a different world with different values, and the target functions for which the claimed handoffs form a complete workflow that executes in new worlds.

We then give the claimed graph to a language-model agent and count the target functions it reaches in data that exploration never saw. Our findings are the following.

- Rebuilding inputs, not choosing experiments, limited our first explorer. In the masked condition it reproduced 0.5 of the 24 masked handoffs on average, and its claimed graph completed 5% of target workflows. Refreshing roots and replaying recipes raised these to 24 of 24 and 100% in every run, with no false handoff claimed, at a cost of 19% of the calls.
- Witnessing a handoff and claiming it are different outcomes. In the same condition, refreshing roots alone let the explorer witness 139 of the 144 data handoffs, but its claimed graph completed 12% of workflows: few witnessed handoffs were repeated often enough to pass the claim threshold. Replay supplies the repetitions.
- With independent names, the explorer without reconstruction witnessed 0.4 handoffs on average. Reconstruction let it witness a complete workflow for every target. Learning which output names correspond to which input names, and proposing each learned correspondence at other functions, raised the witnessed data handoffs from 83 to 144, and with rejection memory as well, the claimed graph completed every workflow.
- The claimed graph helps a language-model agent. Given ICHNAEA's graph, the agent reached all 18 targets in each naming condition. With function signatures alone it reached 16 of 18 with real names, 10 of 18 with masked names and none of 12 with independent names, and on the targets it reached either way the graph cut its calls to a third (Section 5.2).
- On AppWorld's nine applications and 457 APIs, all 33 evaluation goals stay reachable without any handoff that neither names nor documentation reveal, under our definitions of both, and an agent that reads the documentation reaches the goal in 95 of 99 attempts over three task worlds. Goal reach there therefore cannot show what exploration adds. AppWorld does show a second limit: a review rating of 4 passes as a song ID because song 4 exists, so a successful call establishes acceptance, not meaning (Section 5.3).

The procedural results rely on identifiers that rarely collide, error messages that name every rejected argument, and parameter names that recur across functions. We state which conclusions depend on which of these.

2 Related work

Learning tool behaviour. STE (Wang et al., 2024a) learns from imagined scenarios and tool calls; DRAFT (Qu et al., 2025), PLAY2PROMPT (Fang et al., 2025) and ToolObserver (Hallinan et al., 2026) use interaction to improve tool descriptions. In-N-Out (Lee et al., 2025) represents output-to-input connections as a graph and predicts them from documentation. ICHNAEA learns parameter-level connections by calling functions before any task, and runs without a language model.

Inferring API dataflow. API testing tools such as RESTler (Atlidakis et al., 2019), Morest (Liu et al., 2022) and AutoRestTest (Kim et al., 2025) use producer–consumer dependencies and execution feedback to reach deeper API states. APIPilot (Nguyen et al., 2026) separates predicted dependencies from execution evidence. Their aim is to find bugs; ours is to learn reusable handoffs within a fixed call budget. Value containment draws on inclusion-dependency discovery (Bauckmann et al., 2007; Papenbrock et al., 2015).

Experience before tasks. Voyager (Wang et al., 2023) stores skills, Agent Workflow Memory (Wang et al., 2024b) extracts workflows from trajectories, and PREPING (Choi et al., 2026) builds memory by solving practice tasks. ICHNAEA stores recipes for a different purpose: to rebuild the inputs of further experiments after the data change, while no task has been given.

Choosing experiments. Expected information is a standard criterion in experimental design (Lindley, 1956; Chaloner & Verdinelli, 1995) and active learning (Houlsby et al., 2011). We use a score built from per-handoff information to decide which untried handoffs to test.

3 SETTING

A tool world offers callable functions, each with named input parameters, and returns responses whose fields carry values. A handoff $e = (p, f, c, q)$ passes the value of field f from producer p to parameter q of consumer c. The explorer sees function signatures and responses, including error messages. It does not see documentation, the evaluator's handoff graph, or any task.

During exploration the explorer keeps four things, each with its own role: (1) the *values* it currently holds, collected from responses in the present state of the data; (2) *recipes*, one per function that has succeeded, recording which function and field supplied each argument; (3) *witnesses*, the handoffs used in some successful call; and (4) *claimed* handoffs, the graph it would hand to a later user. When the data change, it discards the values and keeps the other three.

In the procedural worlds the data change every 500 calls. We evaluate the claimed graph against the generator's handoff graph, which is independent of the explorer. On AppWorld, where the true graph is unknown, we use expert annotations instead, and for authentication we supply one executed login per application. The procedural experiments use no demonstration.

4 ICHNAEA

Beliefs and claims. For each candidate handoff e, ICHNAEA keeps a belief $\mathrm{Beta}(\alpha_e, \beta_e)$ over the probability that a call using e is accepted. The belief starts from a prior with mean m_e and mass κ_e, so that $\alpha_e = m_e\kappa_e$ and $\beta_e = (1 - m_e)\kappa_e$. A success adds one to α_e; a failure adds to β_e the share of blame assigned to e. ICHNAEA claims e once it has at least one success and a posterior mean of at least one half. The prior is fixed when e's first outcome is recorded. Because most tried handoffs fail, the pooled success rate that enters the prior falls as exploration proceeds, so a handoff first tested late can need many successes before its mean reaches one half. Replayed calls are executed calls and update beliefs like any other; a repeated success in a world already visited is not independent evidence that the handoff is meaningful.

Assigning failures. A failed call is evidence against at least one of its handoffs. When the error names the rejected arguments, the failure goes to their handoffs, in full if one is named and shared as below if several are. A named handoff is spared when it already has accepted values and the rejected value is new to it, since the value rather than the handoff may be at fault. When the error names no argument, one unit of failure is shared among the call's handoffs in proportion to $1 - \mu_e$, where μ_e is the posterior mean, so a handoff already well supported absorbs little of it. Two kinds of handoff are then exempt: one given a new value although it already has accepted values, and one given an accepted value after at least two successes.

Prior and candidates. The prior mean combines two quantities. The first is a success rate pooled over all executed handoffs. The second is value containment: the fraction of the values seen in field f of p that also appear among all values returned under fields named q. When f and q share a name, containment is one by construction, so the score combines an exact-name match with a value-overlap test against fields named like q; it gives no signal when no field carries the parameter's name. With c_e the containment and r the pooled rate after n executed outcomes,

$$m_e = \sigma\big(\mathrm{logit}(0.05 + 0.9\, c_e) + \mathrm{logit}(r)\big), \qquad \kappa_e = 2 + 1 + \min(n, 20), \tag{1}$$

where σ is the logistic function; a term and its share of the mass are omitted when it is undefined. To handle names that do not match, ICHNAEA transfers correspondences it has observed: once a value from a field named f has been accepted by a parameter named q, every held source with field name f becomes a candidate for every parameter named q, and the observed success rate of the pairing (f, q) serves as a lower bound on the predicted success of these proposals. Transfer changes which calls are proposed. It does not alter stored beliefs or the claim rule, which change only through the calls it leads to.

Choosing a call. For each function c and each of its required parameters q, ICHNAEA considers the sources already tried for q and 24 held sources sampled in proportion to their pooled success, and picks the one with the highest product of predicted success d_q and information gain g_q, skipping

Algorithm 1. ICHNAEA: learning a handoff graph by calling functions.
Given: callable functions with parameter names; a budget of B calls.
1: recipes $\leftarrow \emptyset$; call every root
2: **while** calls spent $< B$ **do**
3: **if** the data changed: discard held values; call every root; replay recipes in dependency order
4: **if** a control call is queued: run it and continue
5: **for** each function c and required parameter q: choose the unrejected source maximizing $d_q g_q$
6: execute the call with the largest $V(c)$ (Eq. 3)
7: **if** it succeeded: confirm its handoffs; store a recipe for c if none exists; hold the response values
8: **else**: add negative evidence to the handoff the error names; mark that pairing rejected until the data change
9: **return** handoffs with at least one success and posterior mean ≥ 0.5

Figure 2: ICHNAEA. Root and replay calls are charged to the budget B.

pairings that the service rejected in the current state of the data. The gain is computed from executed outcomes only: for this purpose a handoff with s successes and u failures is scored as $\mathrm{Beta}(1+s, 1+u)$, so a confident prior never makes an untried handoff look already tested. The gain of a belief $\mathrm{Beta}(a, b)$ is the expected information that one outcome gives about the acceptance probability θ,

$$g(a,b) = h\Big(\frac{a}{a+b}\Big) - \mathbb{E}_{\theta\sim\mathrm{Beta}(a,b)}\, h(\theta), \tag{2}$$

where h is the binary entropy; the expectation equals $-\frac{a}{a+b}\big(\psi(a+1)-\psi(a+b+1)\big) - \frac{b}{a+b}\big(\psi(b+1)-\psi(a+b+1)\big)$, with ψ the digamma function. When the source holds a value that the parameter has already accepted, ICHNAEA passes it with probability 0.7; otherwise it passes a value from the source not yet tried with that parameter. With $P = \prod_q d_q$ and q^* the parameter with the lowest d_q, the value of calling c is

$$V(c) = \Big(P\sum_q g_q + (1-P)\, g_{q^*} + \lambda P\, \nu(c)\Big)\, a(c)\, s(c), \tag{3}$$

where the first two terms are the expected information from success and from a failure attributed to the weakest binding, $\nu(c) = 1/(1+n_c)$ rewards functions with few successes n_c, $\lambda = 0.2$, and $a(c)$ and $s(c)$ discount functions that fail for reasons unrelated to their arguments and call signatures that have repeatedly returned no evidence. ICHNAEA executes the call with the largest $V(c)$.

Probes of specificity. After a successful call, ICHNAEA queues two kinds of probe for each bound parameter and runs queued probes before choosing a new call. A *control* repeats the call with that argument replaced by a value of the same type from a different held field. It is an ordinary execution of the substituted handoff, and if it succeeds the parameter is marked as accepting unrelated values. Up to three *null* probes replace the argument with a synthetic value of the same type and magnitude that no field produced; they update no belief and estimate how often the parameter accepts arbitrary values. In the experiments reported here, probes cost calls but do not enter the claim rule.

Rebuilding inputs. At the start, and after every change of the data, ICHNAEA calls each root once in a fixed order. It then re-executes every stored recipe whose inputs it holds, repeating until no further recipe can run. Each function is re-executed at most once per change, so a prefix shared by several recipes is rebuilt once. Every one of these calls counts against the budget.

5 EXPERIMENTS

5.1 SYNTHETIC TOOL WORLDS

All results in this subsection come from synthetic tool worlds built by a procedural generator; no real service is involved. Each generated world has 101 functions: six chains of seven stages, with two interchangeable producers at every stage, a session provider whose key every function after the first stage requires, and 16 distractors. Identifiers are large random integers, so an unrelated

value is almost never accepted, and an error names every rejected argument. The data change every 500 calls and the budget is 16,000 calls. We run ten seeds per configuration. A seed changes the functions' order and opaque names, the identifier values and the explorer's random choices, but not the structure of the chains. In the masked condition, the parameter through which the fourth stage of each chain receives its input is renamed at both interchangeable consumers. In the independent condition, every field name and every parameter name is replaced by an unrelated token; a name keeps one token wherever it recurs as a field and another wherever it recurs as a parameter.

We report two different measures of the result. *Witnessed closure* is the share of targets reachable through handoffs used in some successful call, whether or not they were claimed. *Claimed-graph reuse* is the share of targets for which the claimed handoffs, restricted to true ones, form a workflow that executes in three worlds not seen during exploration. With no false claims, as here, the restriction removes nothing.

Table 1 reports every combination of the three mechanisms in each naming condition, and Figure 3 follows discovery over calls at the masked boundary. Refreshing roots costs 928 calls, 5.8% of the budget; with replay, reconstruction takes between 11% and 20%, depending on how many recipes the explorer has stored. No claimed graph in any run contained a false handoff.

Real names. With real names, the explorer without reconstruction witnessed 110.8 of the 144 data handoffs, and its claimed graph completed 73% of workflows. Refreshing roots alone let it witness all 144 and complete every workflow. Names already identify every handoff in this condition; what the explorer lacked was inputs.

Masked boundary. Without reconstruction, the explorer witnessed on average 58.7 of the 144 true data handoffs, reproduced 0.5 of the 24 masked handoffs in a second world, and reached a claimed-graph reuse of 0.05. Refreshing roots raised the witnessed handoffs to 139.0 and the reproduced masked handoffs to 13.7, but reuse only to 0.12. Refreshing roots and replaying recipes witnessed all 144, reproduced all 24, and reached reuse of 1.0 in every run. Figure 3 shows the order of events. With replay, the explorer had witnessed all 24 masked handoffs by 8,000 calls in every run; claims and reuse followed later, as re-executed recipes accumulated the repeated successes that the claim threshold requires. Rejection memory and transfer changed little once replay was present.

The last row of Table 1 removes the data changes altogether. The explorer without reconstruction then witnessed all 144 data handoffs with real and with masked names, and completed 100% and 77% of workflows. Its failure under changing data therefore comes from the changes rather than from the names. With independent names it still claimed nothing.

Independent names. Transfer is not needed for discovery to occur: refreshing roots and replaying recipes without transfer witnessed 83.0 data handoffs and a complete workflow for every target in every run, but few of them passed the claim threshold (reuse 0.02). Transfer raised the witnessed data handoffs to 144.0, and rejection memory stopped the explorer from retesting pairings already rejected in the same state of the data. With roots, replay, rejection memory and transfer, the explorer claimed 72.0 data handoffs and reached reuse of 1.0. Seventy-two claims suffice because every input has two interchangeable producers.

Held-back seeds and random selection. After these results we ran ten further seeds, not used during development, for three configurations in each naming condition (Table 2). The full method, with roots, replay, rejection memory and transfer, reached reuse of 1.0 and precision of 1.0 in every run, and reproduced all 24 masked handoffs in every masked run. The explorer without reconstruction behaved as before. Choosing calls at random, with every other mechanism unchanged, witnessed at most six data handoffs in any run and completed no workflow, even with real names. Reconstruction is therefore necessary but not sufficient: the belief-guided choice of calls is what finds the handoffs that reconstruction then preserves.

5.2 An agent given the claimed graph

The claimed graph is meant for a later user. We test whether it helps a language-model agent (Gemini 3.5 Flash) reach target functions in data it has not seen. The agent receives the function signatures in the condition's vocabulary and a goal: one successful call to a target function. It writes

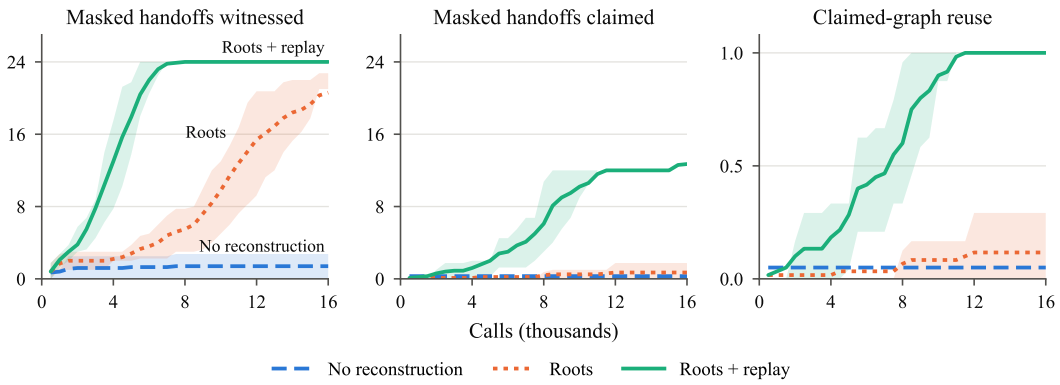


Figure 3: Discovery over calls at the masked boundary (ten runs; lines are means, bands span the 25th to 75th percentiles). Left: masked handoffs witnessed by a successful call. Middle: masked handoffs claimed. Right: share of targets whose claimed handoffs form a workflow that executes in new worlds. Each masked input has two interchangeable producers, so claiming 12 of the 24 masked handoffs suffices for every workflow.

Table 1: Discovery in procedural tool worlds (16,000 calls, means over ten runs). *Witnessed*: true data handoffs, of 144, used in some successful call. *Reproduced*: masked handoffs, of 24, that succeeded in at least two different worlds. *Reuse*: share of targets whose claimed handoffs form a workflow that executes in new worlds. *Rej.*: rejection memory. In the last row the data never change, so reproduction is undefined.

			Real names		Masked			Independent	
Reconstruction	Rej.	Transfer	Witn.	Reuse	Witn.	Repr.	Reuse	Witn.	Reuse
None			110.8	0.73	58.7	0.5	0.05	0.4	0.00
None		✓	108.3	0.67	61.4	1.2	0.03	1.9	0.00
None	✓		111.8	0.63	73.6	1.0	0.05	9.8	0.00
None	✓	✓	105.3	0.65	86.5	3.5	0.07	21.4	0.00
Roots			144.0	1.00	139.0	13.7	0.12	57.6	0.00
Roots		✓	144.0	1.00	143.8	22.1	0.13	143.3	0.00
Roots	✓		144.0	1.00	138.9	11.2	0.13	65.6	0.00
Roots	✓	✓	144.0	1.00	144.0	22.8	0.05	144.0	0.00
Roots + replay			144.0	1.00	144.0	24.0	1.00	83.0	0.02
Roots + replay		✓	144.0	1.00	144.0	24.0	1.00	144.0	0.72
Roots + replay	✓		144.0	1.00	144.0	24.0	1.00	79.2	0.27
Roots + replay	✓	✓	144.0	1.00	144.0	24.0	1.00	144.0	1.00
None, no data change			144.0	1.00	144.0	—	0.77	64.2	0.00

Python that calls the functions, reads their responses and the rejected arguments of failed calls, and has 30 turns and 60 function calls per goal. It sees what the explorer sees and nothing more; the synthetic worlds have no documentation. We use the six targets of each of three held-back seeds (10 to 12) in evaluation world 100001, which no exploration visited. The agent is given no graph, the graph claimed by the explorer without reconstruction, or the graph claimed by ICHNAEA with roots, replay, rejection memory and transfer, each from the held-back run with the same seed and listed per consumer in the agent's vocabulary (Appendix F).

Table 3 gives the results. With ICHNAEA's graph the agent reached every target in every naming condition. With signatures alone, its success fell as names revealed less: 16 of 18 targets with real names, 10 of 18 with masked names and none of 12 with independent names. The graph claimed without reconstruction helped where names still carried cues and not otherwise. With independent names that explorer claimed nothing, so the agent received an empty graph and reached no target in 18 episodes. On the targets reached both with ICHNAEA's graph and without a graph, 16 with real names and 10 with masked names, the graph reduced the mean number of calls from 24.5 to 8.2 and from 29.4 to 10.0.

Table 2: Replication on ten seeds not used during development (seeds 10 to 19; 16,000 calls, means over ten runs, with the range over runs for reuse), with a control that chooses calls at random. Full: roots, replay, rejection memory and transfer. Random: the same, with each call and each source chosen uniformly at random. Witn.: true data handoffs witnessed, of 144. Repr.: masked handoffs reproduced, of 24.

	Real names		Masked			Independent	
Configuration	Witn.	Reuse	Witn.	Repr.	Reuse	Witn.	Reuse
No reconstruction	110.7	0.77 (0.50–1.00)	55.0	0.2	0.02 (0.00–0.17)	0.2	0.00 (0.00–0.00)
Full	144.0	1.00 (1.00–1.00)	144.0	24.0	1.00 (1.00–1.00)	144.0	1.00 (1.00–1.00)
Full, random selection	1.7	0.00 (0.00–0.00)	1.2	0.0	0.00 (0.00–0.00)	1.9	0.00 (0.00–0.00)

Table 3: A language-model agent reaching target functions in an unseen data world, by the graph it is given: targets reached of episodes run, with the mean number of function calls per reached target in parentheses (at most 60 per goal). Three held-back seeds with six targets each. None: signatures only. No reconstruction: the graph claimed by the explorer without roots or replay. ICHNAEA: roots, replay, rejection memory and transfer. †That explorer claimed no handoff, so the graph is empty. ‡Two of the three seeds, because of our API budget.

Graph given	Real names	Masked	Independent
None	16/18 (24.5)	10/18 (29.4)	0/12‡
No reconstruction	18/18 (9.7)	15/18 (16.1)	0/18†
ICHNAEA	18/18 (8.2)	18/18 (9.1)	18/18 (10.2)

5.3 APPWORLD

AppWorld has nine applications and 457 APIs, with state that persists across calls. We call a handoff opaque when neither a name rule nor the documentation-derived graph implies it. With every opaque handoff removed from the expert-endorsed graph, all 33 evaluation goals remain reachable, so goal reach on AppWorld cannot show what exploration adds beyond names and documentation. We use it instead to test whether the mechanisms behave as intended on a real, stateful suite.

We ran ICHNAEA on the Spotify application for 8,000 calls, with the data changing every 500 calls and ten seeds per configuration (Appendix E). Without reconstruction, a fixed planner given the claimed graph reached 63% of goals through routes that the expert annotation endorses and 16% through routes it does not. With roots and replay, endorsed reach fell to 31% and unendorsed reach rose to 44%; the ranges over seeds do not overlap. Claimed handoffs rose from 490 to 873, and those absent from the annotation from 182 to 609. Adding transfer gave similar results. On all nine applications, with five seeds per configuration, the shift was larger: endorsed reach fell from 60% to 9% and unendorsed reach rose from 7% to 32%. Reconstruction took 20% of the budget there; the explorer witnessed 209 annotated strong handoffs instead of 498, and reproduced 175 instead of 55 in a second task world.

The cause is a second limit that AppWorld exposes. A successful call shows that a value was accepted, not that the handoff is meaningful: a playlist review's `rating` of 4 is accepted as a `song_id` because song 4 exists. Small identifiers of this kind exist in every task world, so a recipe that succeeded by coincidence succeeds again when it is replayed, and the repeated successes carry it past the claim threshold. In the synthetic worlds, identifiers rarely collide, and replay repeats only true handoffs. Where they do collide, reconstruction needs a claim rule that uses evidence beyond acceptance, such as the specificity probes, which are recorded but not used here, or agreement between a response and the record its input came from.

We also gave the all-application graphs to the agent of Section 5.2, which on AppWorld can read the documentation through AppWorld's own documentation functions and starts without credentials. Over the 33 goals in three unseen task worlds, it reached the goal in 95 of 99 attempts with documentation alone, in 98 with the graph claimed without reconstruction, in 96 with the graph claimed with roots, replay and transfer, and in 99 with the documentation-derived graph. Documentation already suffices for nearly every goal, so these goals cannot measure what a learned graph adds for an agent on AppWorld.

6 DISCUSSION

Calls made before any task can reveal handoffs that later execute as complete workflows in worlds the explorer has not seen. In our experiments, the main obstacle to this was not the choice of experiment but the loss of the inputs that experiments need when the data change. Stored recipes remove that obstacle, and they let the explorer resolve handoffs that its names do not reveal. Where identifiers collide, as on AppWorld, the same replays repeat coincidental successes, and the claim rule then admits handoffs that are not meaningful.

Witnessing a handoff and acquiring it as reusable knowledge are different outcomes. With independent names, the witnessed handoffs formed a complete workflow for every target while the claimed graph supported almost none, until transfer and rejection memory were added.

The claimed graph also serves a different kind of user. A language-model agent given it reached targets that it did not reach from names alone, in data that neither the explorer nor the agent had seen, and used a third of the calls on targets it reached either way. What the explorer learned before any task was therefore usable by an agent that did not learn it.

Limitations. Our evidence for the method comes from synthetic worlds with one chain structure, identifiers that rarely collide, error messages that name every rejected argument, parameter names that recur across functions, and no state beyond the periodic data changes; the main numbers come from ten development seeds and are confirmed on ten held-back seeds. Rejection memory is sound only when a rejected source field never supplies a valid value, which holds in these worlds but not in general, and transfer relies on recurring names. The claim threshold is not a calibrated probability that a handoff is meaningful, and replay can repeat a success in a world already visited. Reconstruction is eager: it rebuilds every recipe after each change, at 6–20% of the budget, rather than only the inputs that a chosen experiment needs. Random selection under the same reconstruction fails, but we have not separated the contribution of the information term from that of the prior's prediction of success. On AppWorld, replay turned coincidental successes into claims, so reconstruction there needs a claim rule that uses evidence beyond acceptance. We have not compared ICHNAEA with language-model agents that explore the same worlds themselves. The benefit to an agent is shown with one model, for reaching target functions in the synthetic worlds; on AppWorld, documentation alone let the agent reach nearly every goal, and we did not evaluate AppWorld's tasks.

ETHICS STATEMENT

This work involves no human subjects and no personal data. The synthetic tool worlds are generated, and AppWorld's applications, users and records are simulated; every tool call in our experiments, including the language-model agent's, went to these local environments. Exploration by calling tools has side effects: on real services it can create, change or delete records, send messages or incur costs, and calling a service's functions systematically resembles probing it. An explorer such as ICHNAEA should therefore run in a sandbox or on a copy of the data, and only on tools one is authorised to use.

REPRODUCIBILITY STATEMENT

Appendix A specifies ICHNAEA completely, including every constant; Appendix B gives the generator of the synthetic tool worlds, which produces each world deterministically from its seed; Appendix C defines every measure; and Appendices E and F give the AppWorld configuration and the agent protocol. The explorer never reads the ground truth, which only the evaluator uses. Each run records its configuration, the hashes of its source files and a ledger of every call, and is accepted only if the ledger has one row per charged call. An interrupted and resumed run produces the same ledger and graph as an uninterrupted one. With every mechanism off, the explorer reproduces the call ledgers of the earlier implementation byte for byte, in all twenty real-name and independent-name runs and all five AppWorld runs of the earlier study, so differences between configurations come from the mechanisms rather than from reimplementation. The agent study depends on a hosted model, whose replies may change between versions. The code, the generator, and the analysis and figure scripts are provided as supplementary material.

AI USE STATEMENT

In this work, we used generative AI tools (Claude Opus 5, Gemini 3.1 Pro) to edit the manuscript for readability, grammar, and clarity, and to assist with identifying and summarizing related literature. We also used AI coding assistants to help write and debug parts of the experimental codebase, including the generator of the synthetic tool worlds, which then produces every world deterministically from a seed. All research ideas, problem formulations, and experimental design are the authors' own. All AI-assisted work was reviewed by the authors.

REFERENCES

Vaggelis Atlidakis, Patrice Godefroid, and Marina Polishchuk. RESTler: Stateful REST API fuzzing. In *Proceedings of the 41st International Conference on Software Engineering (ICSE)*, 2019.

Jana Bauckmann, Ulf Leser, Felix Naumann, and Veronique Tietz. Efficiently detecting inclusion dependencies. In *IEEE International Conference on Data Engineering*, 2007.

Kathryn Chaloner and Isabella Verdinelli. Bayesian experimental design: A review. *Statistical Science*, 10(3):273–304, 1995.

Yumin Choi, Sangwoo Park, Minki Kang, Jinheon Baek, and Sung Ju Hwang. PREPING: Building agent memory without tasks. *arXiv preprint arXiv:2605.13880*, 2026.

Wei Fang, Yang Zhang, Kaizhi Qian, James Glass, and Yada Zhu. PLAY2PROMPT: Zero-shot tool instruction optimization for LLM agents via tool play. In *Findings of the Association for Computational Linguistics: ACL 2025*, 2025.

Skyler Hallinan, Thejas Venkatesh, Xiang Ren, Sai Praneeth Karimireddy, Ashwin Paranjape, Yuhao Zhang, and Jack Hessel. OpaqueToolsBench: Learning nuances of tool behavior through interaction. *arXiv preprint arXiv:2602.15197*, 2026.

Neil Houlsby, Ferenc Huszár, Zoubin Ghahramani, and Máté Lengyel. Bayesian active learning for classification and preference learning. *arXiv preprint arXiv:1112.5745*, 2011.

Myeongsoo Kim, Tyler Stennett, Saurabh Sinha, and Alessandro Orso. A multi-agent approach for REST API testing with semantic graphs and LLM-driven inputs. In *Proceedings of the 47th IEEE/ACM International Conference on Software Engineering (ICSE)*, 2025.

Seungkyu Lee, Nalim Kim, and Yohan Jo. In-N-Out: A parameter-level API graph dataset for tool agents. *arXiv preprint arXiv:2509.01560*, 2025.

D. V. Lindley. On a measure of the information provided by an experiment. *The Annals of Mathematical Statistics*, 27(4):986–1005, 1956.

Yi Liu, Yuekang Li, Gelei Deng, Yang Liu, Ruiyuan Wan, Runchao Wu, Dandan Ji, Shiheng Xu, and Minli Bao. Morest: Model-based RESTful API testing with execution feedback. In *Proceedings of the 44th International Conference on Software Engineering (ICSE)*, 2022.

Tu Nguyen, Thanh Nguyen, Huy Nguyen, Viet Nguyen, Tien N. Nguyen, and Vu Nguyen. Rest api testing with verified llm-inferred dependencies and response-driven refinement, 2026. URL `https://arxiv.org/abs/2608.17546`. Preprint.

Thorsten Papenbrock, Sebastian Kruse, Jorge-Arnulfo Quiané-Ruiz, and Felix Naumann. Divide & conquer-based inclusion dependency discovery. *Proceedings of the VLDB Endowment*, 8(7): 774–785, 2015.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. From exploration to mastery: Enabling LLMs to master tools via self-driven interactions. In *International Conference on Learning Representations*, 2025.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024.

Boshi Wang, Hao Fang, Jason Eisner, Benjamin Van Durme, and Yu Su. LLMs in the imaginarium: Tool learning through simulated trial and error. *arXiv preprint arXiv:2403.04746*, 2024a.

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.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024b.

A ICHNAEA IN FULL

Interface. The explorer sees each function's name and the names of its parameters, marked required or optional, and each call's outcome: a response flattened into named fields with their values, or an error with its kind and, where the service states it, the rejected arguments. It never binds optional parameters.

State. Four structures are kept apart. (1) Held values: for each producer and field, up to 16 distinct values from responses in the present state of the data, the oldest dropped first; all are discarded when the data change. (2) Recipes: for each function, the first plan with which it succeeded, recording the source function and field of every argument. Plans of control calls are not stored, and on AppWorld neither are those of the login demonstration. (3) Beliefs: one Beta distribution per handoff that has an outcome, with pooled statistics. (4) Rejections: the pairings rejected by an error in the present state of the data, cleared when it changes.

Beliefs. A handoff's prior is set by Eq. 1 when its first outcome is recorded and is not revised afterwards. The pooled rate r is the posterior mean of a single $\mathrm{Beta}(1,1)$ pool updated by every outcome of every handoff; containment is computed from the values collected in the present state of the data, capped at 64 per field and 256 per field name. A success adds one to α_e for every handoff the call used. Failures are assigned as described in Section 4. On AppWorld, an error reporting a conflict with the current state, such as an item already liked, or with a record's owner, adds one half to α_e for handoffs not yet confirmed this way, since the service accepted the arguments. A handoff is claimed when its success mass is at least one and its posterior mean at least one half.

Choosing a call. Each step first runs one queued null probe, or, if none is queued, one queued control; otherwise it chooses a new call. For every function with required parameters and every such parameter q, the candidate sources are the held sources already tried for q, 24 held sources drawn with replacement in proportion to their pooled success rate plus 0.05, and, with transfer, every held source whose field name has a witnessed correspondence with q's name. With rejection memory, pairings rejected in the present state are removed. The chosen source maximizes $d_q g_q$, where d_q is the posterior mean, raised with transfer to the correspondence's success rate when that is higher, and g_q is Eq. 2 applied to $\mathrm{Beta}(1{+}s, 1{+}u)$. A function is skipped when some parameter has no candidate left. A function without required parameters is scored as $g + \lambda d\, \nu(c)$, where d and g come from a belief over whether calling it succeeds. Every score is multiplied by $a(c) = (n_c^+ + 1)/(n_c + 1)$, where n_c counts c's calls and n_c^+ those that did not fail for reasons outside the arguments, and by $s(c) = 1/(1+k)$, where k counts earlier calls with the same function and bindings that returned no evidence. Ties are broken at random.

Reconstruction. At the start and after each change of the data, ICHNAEA calls every function without required parameters in lexicographic order of their names. With replay, it then passes repeatedly over its recipes in the order they were stored and executes each recipe whose sources all hold values, choosing values as in an ordinary call, until a pass executes nothing. A function is re-executed at most once per change.

Constants. All experiments use the same values, fixed before the experiments reported here: prior mass $\kappa_0 = 2$, pooled mass capped at 20, unit weights on containment and pooled rate, 24 sampled candidates, $\lambda = 0.2$, reuse probability 0.7, 16 held values per field, three null probes and one control per parameter, and a change of data every 500 calls. The budget is 16,000 calls in the synthetic worlds and 8,000 on AppWorld. Every call counts against it, including probes, root and replay calls and, on AppWorld, the login demonstration.

B SYNTHETIC TOOL WORLDS

Functions. A session provider takes no input and returns a session key. Each of six chains has seven stages, numbered 0 to 6, and each stage has two interchangeable functions. A stage-0 function takes no input and returns a list of eight records. A function at stage $k \geq 1$ of chain j takes the session key and the identifier field of stage $k-1$, named `resource`j`_stage(`$k-1$`)_id`, and returns the record whose identifier it was given, with that record's stage-k identifier. Every record also carries a fingerprint. Sixteen distractors take no input and return a rating between 1 and 8 and a fingerprint. Function names are opaque hashes, and their order in the interface is shuffled by the seed. The world has 101 functions.

Values and errors. Identifiers are integers derived from a hash of the seed, world, chain, stage and record, above 10^3 and below about 4.5×10^{15}, so an unrelated value is accepted with negligible probability. A call with a wrong session key or an identifier outside the current records fails with a validation error that names every rejected argument. A change of data moves the world to the next of eight world IDs, cycling, which changes every identifier, fingerprint and session key; the functions and the handoff graph stay the same. Worlds 100001 to 100003 are used only for evaluation.

Truth. Each of the 72 functions after stage 0 has three true handoffs: the session key and its identifier from either of the two functions of the previous stage. This gives 216 handoffs, 144 of them data handoffs. The 24 data handoffs into stage 3 form the masked boundary. The targets are one stage-6 function per chain. The generator checks that removing the boundary handoffs leaves every target unreachable and that the full graph reaches all of them.

Vocabularies. With real names, the interface is as above. In the masked condition, the six stage-3 parameter names are replaced by opaque tokens; all other names are unchanged. In the independent condition, every application, function, field and parameter name is replaced by a token derived from a hash of the name, its role and the seed, so a name keeps one token as a field and a different token as a parameter. Error messages are suppressed in every condition; the rejected arguments are reported in the explorer's vocabulary. Before any call, a check confirms the intended cue structure for every seed: exact name matching reaches all six targets with real names, none in the masked condition (it reaches 53 of the 101 functions, the first three stages and the roots) and none with independent names.

C MEASURES

Ground truth enters only these computations, never the explorer. A data handoff is *witnessed* once a call using it succeeds, in any kind of call. A masked handoff is *reproduced* when calls using it succeed in at least two different world IDs, and therefore with different identifiers; an earlier version of this count used reset phases instead of world IDs, which counts a revisited world twice and changes the no-reconstruction mean from 0.6 to 0.5. *Claimed* handoffs pass the claim rule of Appendix A at the end of the budget, and *precision* is the share of claimed handoffs that are true; it is undefined when nothing is claimed. *Witnessed closure* is the share of targets reachable through witnessed true handoffs, with every required parameter supplied. *Reuse* executes, in each of three evaluation worlds, a backward-built route to every target through the claimed handoffs restricted to true ones, and counts the targets whose final call succeeds. In these worlds a true route always executes, so reuse equals the share of targets reachable through the claimed true handoffs; with no false claims, as in every run here, the restriction removes nothing. *Input availability*, used in the introduction, is the share of exploratory calls with arguments made while the session key and the

required identifier had been produced in the current state of the data. *Reconstruction cost* counts root and replay calls.

D FULL RESULTS

Tables 4, 5 and 6 extend Table 1 with every measure, for each naming condition. The held-back replication is in Table 2.

Table 4: All configurations with real names (16,000 calls, means over ten runs). Avail.: share of exploratory calls with arguments whose required inputs were available. Witn. and Claimed: true data handoffs, of 144. Prec.: precision of the claimed graph (— when nothing is claimed). W. clos.: witnessed closure. Recon.: root and replay calls, as a share of the budget.

Reconstruction	Rej.	Tr.	Avail.	Witn.	Claimed	Prec.	W. clos.	Reuse	Recon.
None			0.014	110.8	100.9	1.00	0.80	0.73	0.0%
None		✓	0.012	108.3	89.4	1.00	0.75	0.67	0.0%
None	✓		0.013	111.8	93.9	1.00	0.80	0.63	0.0%
None	✓	✓	0.013	105.3	89.9	1.00	0.75	0.65	0.0%
Roots			0.052	144.0	144.0	1.00	1.00	1.00	5.8%
Roots		✓	0.078	144.0	144.0	1.00	1.00	1.00	5.8%
Roots	✓		0.057	144.0	144.0	1.00	1.00	1.00	5.8%
Roots	✓	✓	0.083	144.0	144.0	1.00	1.00	1.00	5.8%
Roots + replay			0.999	144.0	144.0	1.00	1.00	1.00	19.6%
Roots + replay		✓	1.000	144.0	144.0	1.00	1.00	1.00	19.6%
Roots + replay	✓		0.998	144.0	144.0	1.00	1.00	1.00	19.6%
Roots + replay	✓	✓	1.000	144.0	144.0	1.00	1.00	1.00	19.6%
None, no data change			0.974	144.0	144.0	1.00	1.00	1.00	0.0%

Table 5: All configurations with the masked boundary (16,000 calls, means over ten runs). Avail.: share of exploratory calls with arguments whose required inputs were available. Witn. and Claimed: true data handoffs, of 144. Repr.: masked handoffs, of 24, succeeding in two or more worlds. Prec.: precision of the claimed graph (— when nothing is claimed). W. clos.: witnessed closure. Recon.: root and replay calls, as a share of the budget.

Reconstruction	Rej.	Tr.	Avail.	Witn.	Claimed	Repr.	Prec.	W. clos.	Reuse	Recon.
None			0.012	58.7	56.6	0.5	1.00	0.18	0.05	0.0%
None		✓	0.012	61.4	52.0	1.2	1.00	0.17	0.03	0.0%
None	✓		0.014	73.6	61.0	1.0	1.00	0.42	0.05	0.0%
None	✓	✓	0.015	86.5	67.3	3.5	1.00	0.43	0.07	0.0%
Roots			0.087	139.0	119.1	13.7	1.00	0.98	0.12	5.8%
Roots		✓	0.071	143.8	122.2	22.1	1.00	1.00	0.13	5.8%
Roots	✓		0.078	138.9	120.5	11.2	1.00	1.00	0.13	5.8%
Roots	✓	✓	0.073	144.0	121.1	22.8	1.00	1.00	0.05	5.8%
Roots + replay			0.851	144.0	132.7	24.0	1.00	1.00	1.00	18.6%
Roots + replay		✓	0.896	144.0	144.0	24.0	1.00	1.00	1.00	18.9%
Roots + replay	✓		0.850	144.0	132.7	24.0	1.00	1.00	1.00	18.5%
Roots + replay	✓	✓	0.862	144.0	144.0	24.0	1.00	1.00	1.00	18.6%
None, no data change			0.869	144.0	131.9	—	1.00	1.00	0.77	0.0%

E APPWORLD STUDY

Configuration. The explorer runs on AppWorld's Spotify application with real names, for 8,000 calls, in ten seeds for each of three configurations: no reconstruction; roots and replay; roots, replay and transfer. The same configurations also run on all nine applications, in five seeds each. The data change every 500 calls, cycling through eight training task worlds, and one login demonstration

Table 6: All configurations with independent names (16,000 calls, means over ten runs). Avail.: share of exploratory calls with arguments whose required inputs were available. Witn. and Claimed: true data handoffs, of 144. Prec.: precision of the claimed graph (— when nothing is claimed). W. clos.: witnessed closure. Recon.: root and replay calls, as a share of the budget.

Reconstruction	Rej.	Tr.	Avail.	Witn.	Claimed	Prec.	W. clos.	Reuse	Recon.
None			0.004	0.4	0.0	—	0.00	0.00	0.0%
None		✓	0.004	1.9	0.0	—	0.00	0.00	0.0%
None	✓		0.007	9.8	0.0	—	0.00	0.00	0.0%
None	✓	✓	0.009	21.4	0.0	—	0.00	0.00	0.0%
Roots			0.079	57.6	0.0	—	0.30	0.00	5.8%
Roots		✓	0.080	143.3	0.0	—	0.98	0.00	5.8%
Roots	✓		0.067	65.6	0.0	—	0.48	0.00	5.8%
Roots	✓	✓	0.064	144.0	0.0	—	1.00	0.00	5.8%
Roots + replay			0.474	83.0	6.8	1.00	1.00	0.02	11.3%
Roots + replay		✓	0.579	144.0	56.0	1.00	1.00	0.72	14.3%
Roots + replay	✓		0.646	79.2	33.8	1.00	0.98	0.27	14.2%
Roots + replay	✓	✓	0.727	144.0	72.0	1.00	1.00	1.00	16.6%
None, no data change			0.284	64.2	0.0	—	0.78	0.00	0.0%

is charged at the start and after every change. Two choices were fixed before any run. Rejection memory is off, because on AppWorld a rejected value does not show that its source field never supplies a valid one: validity depends on the value, its owner and the current state. The login demonstration is not stored as a recipe, because it is repeated after every change anyway. With no reconstruction, the explorer is the one of the earlier AppWorld study; its ledgers match that study's byte for byte for the five seeds it ran.

Scoring. Claims are compared with the expert annotation, excluding demonstrated handoffs. Each final graph is then given to a fixed planner in three development worlds not used during discovery, with 150 calls per target, depth 6, twelve value combinations per parameter and at most 8 or 16 candidate sources per parameter.

Route recovery and endorsement. The planner records every successful call it makes while searching for a goal, including detours it abandons. The *route* of a completed goal is recovered from the stored provenance of each call, which names, for every argument, the function and field that supplied it. Starting from the goal call, each argument's source is resolved to the most recent earlier call of the producing function; the process repeats from those calls until no new call is added. A route is *endorsed* if each of its handoffs is a strong or weak edge of the expert annotation or a handoff some human solution performs. The annotation is used by the evaluator only. The rule can miss sound unannotated handoffs and can endorse a route that uses the wrong record for a task; we therefore report reach with and without endorsement, not task success.

Results. Table 7 gives the results. *Reproduced* counts annotated strong handoffs, other than demonstrated ones, that succeeded in at least two different task worlds. Reach is measured at 8 candidate sources per parameter; at 16 the values differ by at most 0.03. For scale, the same planner reaches 81% of Spotify goals with the expert-annotated graph or the documentation-derived graph with credentials, and 80% and 76% of the goals on all applications; these are references, not comparison targets, since their reach depends on planner capacity.

F AGENT STUDY

Synthetic worlds. The agent is Gemini 3.5 Flash at temperature 0, asked for a JSON reply with a one-sentence thought and the Python code for the turn. Each prompt contains the instructions, every function with its required parameters in the condition's vocabulary, the goal, the graph if one is given, and the code and output of the last 14 turns. The code runs with a restricted set of built-ins and no imports, under a limit of 10 seconds per turn, and its variables persist between turns. A failed

Table 7: AppWorld (8,000 calls; ten runs on Spotify, five on all nine applications; means, with the range over runs for reach). Recall: annotated strong handoffs claimed, excluding demonstrations. Unannot.: claimed handoffs absent from the annotation. Non-edges: claimed handoffs the annotation labels as non-edges. Recon.: root and replay calls as a share of the budget. Endorsed and unendorsed: goals reached by the fixed planner in three unseen task worlds.

Configuration	Recall	Claimed	Unannot.	Non-edges	Repr.	Recon.	Endorsed	Unendorsed
Spotify								
No reconstruction	0.244	490.2	181.6	0.1	83.6	0.0%	0.63 (0.50–0.73)	0.16 (0.08–0.27)
Roots + replay	0.205	872.9	608.6	4.6	82.4	13.3%	0.31 (0.19–0.44)	0.44 (0.29–0.52)
Roots + replay + transfer	0.227	859.9	567.9	4.3	77.8	13.1%	0.24 (0.12–0.48)	0.43 (0.23–0.52)
All applications								
No reconstruction	0.034	607.4	57.0	19.2	55.2	0.0%	0.60 (0.54–0.65)	0.07 (0.05–0.10)
Roots + replay	0.014	320.6	64.6	13.0	175.2	20.3%	0.09 (0.07–0.12)	0.32 (0.21–0.37)
Roots + replay + transfer	0.013	282.2	56.0	4.6	163.8	20.0%	0.03 (0.03–0.03)	0.36 (0.27–0.42)

call returns the rejected arguments, as the explorer would see them. An episode ends when the target call succeeds, when 60 calls have been made, or after 30 turns. The graph lists, for each consumer, every claimed handoff as `parameter <- function.field`: 216, 216 and 144 handoffs from ICHNAEA with real, masked and independent names, and between 160 and 194, between 67 and 89, and none from the explorer without reconstruction. A reply that is not valid JSON with code counts as a turn without a call; this happened 10 times, all with masked names and no graph.

Each request may produce up to 65,536 output tokens, the model's reasoning included, and may take up to 600 seconds. An earlier pass with a cap of 8,192 tokens was discarded: on some requests, mostly in the conditions without a graph, the reasoning exhausted the cap and the reply came back empty. In the reported episodes no reply reached the cap. Episodes during which a request to the model failed were discarded and rerun. With independent names and no graph, 12 episodes were run, on seeds 10 and 11, because of our API budget. The synthetic study cost about US$56.

AppWorld. The agent calls AppWorld's APIs through AppWorld's own Python interface, may read the documentation through its documentation functions, and starts without credentials. A goal is one successful call to one of the 33 functions that most often end human solutions, in task worlds 12 to 14, which discovery did not use, with 30 turns and 60 API calls. With a graph, the prompt adds up to 60 bindings, those of the target first and then those of other functions of its application, at most six per function. The graph for task world $12 + k$ is the final all-application graph of discovery seed k, with demonstrated handoffs removed; the documentation-derived graph contains the strong and weak edges that In-N-Out predicts (Lee et al., 2025). The agent reached the goal in 95 of 99 attempts with documentation alone, 98 with the graph claimed without reconstruction, 96 with the graph claimed with roots, replay and transfer, and 99 with the documentation-derived graph. This study cost about US$33. AppWorld's own tasks were not run.