DISCOVER · MIRROR · CAPTURE

DISCOVERY.

MIRROR ANY SYSTEM INTO PROTO.

A self-installing agent that walks into any existing system, observes how it really behaves, captures the rules and workflows that govern it, and replays the whole thing inside ERP.AI as a live Proto. You don't migrate. You discover.

DOCUMENT	VERSION	AUDIENCE	FORMAT
DISC-WP-001	1.0 · April 2026	Architects · CIOs	Letter

ABSTRACT

Discovery is a Proto-class agent that mirrors an existing software system — legacy ERP, custom internal tooling, spreadsheet-driven process, third-party SaaS — into a live ERP.AI Proto without manual migration, without a rip-and-replace project, and without asking the source system to expose anything beyond its normal user-facing surface.

Conventional system migrations begin with a discovery phase that takes weeks to months: business analysts shadow operators, dump schemas, write down "the way we actually do things," and reconcile that against the way the new system wants the work done. The output is documentation. Discovery skips the documentation step. It is the agent that performs the discovery, and its output is a running mirror — the same entities, the same states, the same rules, encoded as Proto policies that ERP.AI can execute.

This whitepaper describes the Discovery agent in operational depth: the four-phase loop (**Discover** → **Observe** → **Capture** → **Replay**), the surface-agnostic ingestion model, the rule-extraction methodology, the handoff contract to Proto, and the safety envelope (read-only by default, human-gated promotions, full replayable trace).

It also presents an evaluation framework (Chapter 7) for measuring Discovery's success against a captured workflow — coverage, fidelity, drift, replay accuracy — and three anonymized case sketches (Chapter 8) that walk through how Discovery handled a major legacy ERP, a Tier-1 CRM, and a spreadsheet-driven approval pipeline.

> *"You don't move the data. You move the behavior. The data follows because the behavior brought it."*

WHO THIS IS FOR

Enterprise architects, CIOs, transformation leads, and engineering directors who are responsible for replacing or augmenting an incumbent system and want a faster path than the eighteen-month implementation plan currently sitting on their desk. Also: Proto operators who want to bring existing institutional knowledge into ERP.AI as policy rather than as documents nobody reads.

WHAT THIS IS NOT

This is not a connector library. Discovery does not assume the source system has an API, a webhook, a documented schema, or a vendor that returns calls. It assumes the source system has a surface a human can use — and reads that surface the way a senior operator would.

HOW TO READ THIS DOCUMENT

Linearly, if you have an hour. If not: the abstract above, the four-phase loop in Chapter 4, and the evaluation framework in Chapter 7 are the load-bearing chapters. Chapter 8's case sketches are the most persuasive if you've lived through a migration. The configuration surface is in Chapter 9, with the full API at /proto/discovery.md.

CONTENTS

STATUS OF EVALUATION DATA

All quantitative figures in Chapters 7 and 8 are clearly marked *illustrative — pre-launch projection*. Discovery is in restricted preview at the time of writing (April 2026); production benchmark data will replace the projections in v1.1 and is governed by the public versioning policy at /changelog.

THE MIGRATION PROBLEM

Every enterprise system replacement begins with a discovery phase whose deliverable is a binder. The binder is wrong before it is printed. The system that produced it kept moving while it was being interviewed.

The textbook playbook for replacing an incumbent system — an ERP, a CRM, a custom internal app, a workflow engine, a department's shared spreadsheet that has accidentally become the source of truth — runs on a predictable cadence: requirements gathering, fit-gap analysis, configuration, parallel run, cutover. The first phase is by far the longest, and its deliverable is documentation. Process diagrams. Approval matrices. Field mappings. A long list of "*this is how it works today, this is how it should work tomorrow.*"

Three things conspire against this approach. First, the documentation is wrong before the analyst stands up from the desk. The actual workflow has variants the operator forgot to mention, edge cases that never came up during the interview, side-channel approvals that happen in chat, decisions that get made by reading a spreadsheet column the schema doesn't describe. Second, the documentation is read once, by the implementation team, and then becomes a fossil. By the time anyone looks at it again, the source system has moved. Third, the documentation cannot be executed. It has to be re-encoded as configuration in the target system — another translation step, another opportunity for behavior to drift from the original.

The cumulative result is the eighteen-to-thirty-month implementation, the parallel-run period that nobody trusts, the cutover that is followed by months of "it's not behaving like the old system" tickets, and the executive summary that uses the word "*transformation*" without irony.

The honest position is that the binder was never the deliverable. The deliverable was always the running mirror — a system that does what the old one does, accurately enough that operators recognize it. The binder was just the project management methodology insisting on an artifact it could review.

The Discovery agent skips the binder. It performs the discovery, captures the rules in a form that ERP.AI can execute, and produces a running Proto on the other side. Two phases that used to be six months become one phase that takes the time it takes — and the human review happens against running behavior rather than against a Visio diagram.

This is not new in concept. *Process mining* tools have spent a decade reconstructing workflows from event logs. What is new is the agentic capability that can *also act* on what it discovers — spawn the entities, encode the rules as policies, hand the running result to Proto, and start operating it.

WHAT WE WILL AND WILL NOT CLAIM

Discovery is not a magic wand. There are systems Discovery will not finish — ones whose behavior is genuinely undocumented, ones whose operators disagree about what the rules even are, ones whose data is so dirty that the mirror has to halt. Chapter 7 presents the failure modes alongside the success cases. The evaluation framework is designed to make those failure modes visible early and let humans intervene before the mirror commits.

DISCOVER. MIRROR. CAPTURE.

Three modes, one mission: hand Proto a faithful clone of how the source system really runs — not how it is documented, not how it is supposed to run, but how it actually behaves on a Tuesday afternoon.

PILLAR 01 · DISCOVER

Crawl the source system end-to-end. Surface every screen, schema, queue, queue dead-letter, integration endpoint, and report. Build a topology of *what exists* before deciding *what matters*. Discovery does not pre-judge scope; it maps the territory and lets the human-approved capture policy decide which regions to dig deeper into.

PILLAR 02 · MIRROR

Replicate the structure inside ERP.AI: the same entities, the same states, the same approval chains, the same field semantics. The mirror starts hollow and fills as Discovery learns. Mirroring is not data copying — it is *shape capture*. Data follows once the shape is correct enough that the source system's records have a place to land.

PILLAR 03 · CAPTURE

Pull the implicit rules: who approves what over which threshold, which exceptions get auto-routed to which queue, the conditions humans actually check before clicking *Approve*. Encode those rules as Proto policies — testable, replayable, version-controlled — and stop relying on the operator's knowledge being on-call.

THE THREE PILLARS RUN CONCURRENTLY

Discover does not "complete" before Mirror starts; Capture does not wait for Mirror to finish. The three pillars share a working memory and update each other: a new entity surfaced by Discover prompts Mirror to allocate shape, which prompts Capture to look for the rules that govern that entity's state transitions. The loop iterates until the success criteria in Chapter 7 are met or until human review halts it.

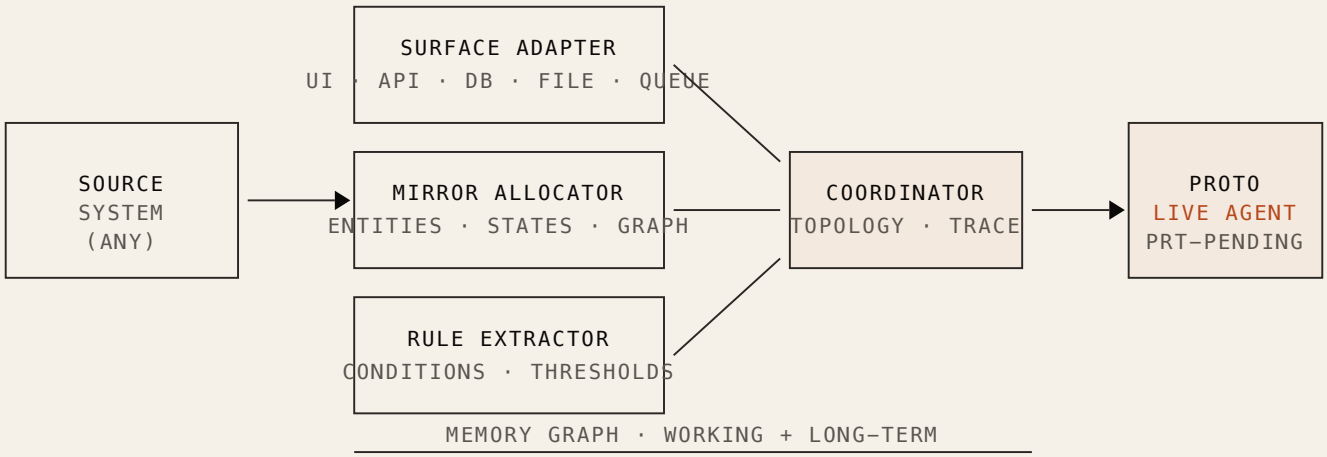
WHAT "LIVE PROTO" MEANS AT HANDOFF

Discovery is finished when Proto can execute the captured workflow end-to-end, on its own data, without human help, on a parallel-run basis. *End-to-end* means a representative cross-section of the workflow's real volume: not the happy path, not the demo case, but a sample drawn from the source system's actual recent history. *Without human help* means no manual intervention during execution, though human-approval gates remain configured for high-stakes actions per the safety policy in Chapter 10.

ARCHITECTURE OVERVIEW

Discovery is a Proto-class agent. It inherits Proto's architectural primitives — self-generating topology, runtime tool synthesis, persistent knowledge graph, multi-agent coordination — and adds three Discovery-specific subsystems: the Surface Adapter, the Mirror Allocator, and the Rule Extractor.

COMPONENT DIAGRAM



SUBSYSTEM RESPONSIBILITIES

SUBSYSTEM	RESPONSIBILITY	OUTPUT
Surface Adapter	Reads whichever surface the source exposes — UI screens, REST endpoints, read-only DB connections, file drops, queue tails. Synthesizes adapter code at runtime if no native client exists.	Normalized event stream
Mirror Allocator	Maintains the shape-capture graph: entity types, fields, states, relationships. Allocates ERP.AI-side equivalents and tracks the source-to-mirror mapping.	Live entity graph
Rule Extractor	Watches state transitions and human decisions in the event stream. Hypothesizes conditions, tests them against held-out events, and promotes confirmed rules to candidate Proto policies.	Candidate policy set
Coordinator	Inherited from Proto. Owns the topology, schedules sub-agents, merges their outputs, and writes the replayable trace.	Trace · DISC-id

MEMORY MODEL

Discovery uses Proto's two-tier memory: **working memory** for the in-flight discovery run (the current topology, the events seen this hour, the in-progress rule hypotheses), and **long-term memory** for cross-run knowledge (which

surface adapter worked for which source pattern, which rule shapes are common in approval workflows, which extraction strategies converged fastest on similar systems).

Long-term memory is what makes the second discovery run on a similar system materially faster than the first. The agent is not starting from zero; it is starting from "we've mirrored a system shaped like this before, here is the topology that worked." Operators can inspect, edit, or wipe long-term memory per tenant.

SUB-AGENT TOPOLOGY

Discovery spawns sub-agents per surface region. A typical mid-complexity discovery run has between four and twelve sub-agents:

- **Crawler agents** — one per major surface region (UI, schema, queue). Parallel.
- **Mirror agents** — one per entity type discovered. Sequential within an entity, parallel across entities.
- **Capture agents** — one per workflow scope (e.g. AP-approval, PO-routing, refund-handling).
- **Verifier agents** — spawned on demand to test hypothesized rules against historical events.

The Coordinator decides when to spawn, when to merge, and when to dissolve. Topology is dynamic — if a Capture agent encounters a workflow region that turns out to be three sub-workflows, the Coordinator splits the agent into three rather than forcing the original to handle all three.

SANDBOX MODEL

Every Discovery run executes inside an isolated sandbox keyed to the discovery_id. The sandbox has read-only credentials by default; write-back to the source system requires an explicit human-approved policy and a separate, scoped write credential. Cancelling a discovery rolls back the sandbox: mirror entities are tombstoned, candidate rules are discarded, and the long-term memory delta is either committed or reverted per the cancellation policy.

TRACE AND REPRODUCIBILITY

Every observation, every hypothesized rule, every candidate-policy promotion, and every Mirror Allocator decision is written to the replayable trace at `/v1/discovery/:id/trace`. The trace is sufficient to reconstruct what Discovery saw, what it decided, and why — both for human review and for re-running the same discovery deterministically against a captured event stream. This is the audit substrate for compliance, not a debug log; it is meant to be readable.

WHAT DISCOVERY IS NOT (ARCHITECTURALLY)

Discovery is not a connector framework. There is no per-vendor adapter to maintain, no SDK to install on the source side, no agent to deploy on the source host. Discovery operates from the same vantage point a senior operator would: it sees what the operator sees, reads what the operator reads, and reasons over the same evidence. This is a deliberate constraint — it keeps Discovery's coverage independent of vendor cooperation, which historically has been the bottleneck for migration tooling.

THE FOUR-PHASE LOOP

Discover → Observe → Capture → Replay. Each phase has a definition of done, a handoff to the next phase, and a set of metrics that determine whether the loop iterates or advances.

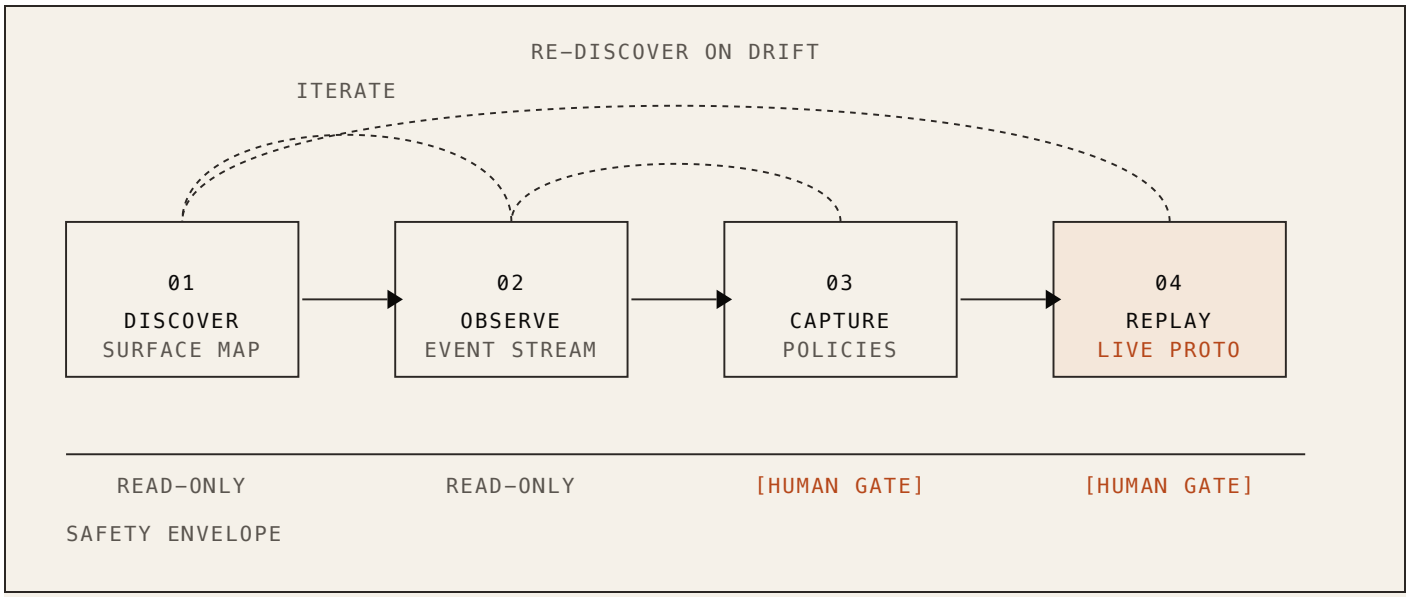
PHASE	WHAT HAPPENS	DONE WHEN	HANDS OFF
01 · Discover	Walk the source system. Map every surface region. Build the entity inventory and the surface-adapter strategy.	Topology coverage exceeds the configured threshold (default 90% of surface regions reachable from a seed) and the inventory is stable across two crawl passes.	Surface map & inventory → Mirror Allocator
02 · Observe	Watch real workflows in motion. Stream events. Capture the variants humans treat as routine, including the off-path ones (Slack approvals, manual overrides, exception queues).	Event stream contains representative samples of every workflow scope in the inventory, weighted by historical volume.	Event stream → Rule Extractor
03 · Capture	Hypothesize rules from observed transitions. Test against held-out events. Promote confirmed rules to candidate Proto policies. Flag ambiguous decisions for human review.	Candidate policies cover the configured fraction of observed decisions (default 95%) at the configured fidelity threshold (default F1 ≥ 0.92 against held-out).	Candidate policy set → Replay
04 · Replay	Run the captured workflow inside Proto on a parallel-run basis using a fresh sample of source events. Compare Proto's decisions against the source system's decisions on the same events.	Replay agreement exceeds the configured threshold (default 95%) and the disagreement set is human-reviewed.	Live Proto (PRT-id)

PHASE DEFINITIONS, EXIT CRITERIA, AND HANDOFFS. DEFAULTS SHOWN ARE THE V1.0 STARTING POINTS; PER-RUN OVERRIDES ARE COMMON.

ITERATION VS. ADVANCEMENT

The loop does not advance monotonically. A Capture phase that fails to reach fidelity threshold can demand more Observe events; an Observe phase that surfaces a new workflow scope can demand a Discover re-pass. Loops are bounded by token budget (Chapter 10) and by a hard ceiling on iteration depth. When a phase exhausts its budget without meeting threshold, the run halts and prompts human review with a summary of what is missing and what would be needed to proceed.

LOOP DIAGRAM



DEFAULT BUDGETS PER PHASE

PHASE	DEFAULT TOKEN BUDGET	WALL-CLOCK TARGET	HARD CEILING
Discover	15,000	15–90 min	3 hr
Observe	25,000	2–72 hr	7 days
Capture	35,000	30–180 min	6 hr
Replay	25,000	30–240 min	8 hr
Total	**100,000**	**3–78 hr**	**~8 days**

DEFAULT BUDGETS FOR A MID-COMPLEXITY DISCOVERY RUN TARGETING A SINGLE WORKFLOW SCOPE. MULTI-SCOPE RUNS SCALE ROUGHLY LINEARLY PER SCOPE, SUB-LINEARLY PER RELATED SCOPE THANKS TO SHARED DISCOVER.

Budgets are configurable per run. The Coordinator emits a budget-warning event when any phase exceeds 75% of its allocation, giving operators a chance to extend rather than halt.

ANY SYSTEM. ANY PROCESS. ANY ENGINE.

Discovery does not maintain a list of supported systems. It maintains a list of *surfaces*: the kinds of evidence a software system can expose to a careful observer. The Surface Adapter recognizes the surface, not the vendor.

SURFACE TAXONOMY

SURFACE	WHAT DISCOVERY DOES WITH IT	TYPICAL EVIDENCE
UI (web)	Headless browser session; reads the rendered DOM, follows links, fills forms, captures screen state and field semantics.	Field labels, validation messages, options in dropdowns, conditional visibility, error states.
UI (desktop)	OS-level accessibility tree where available; OCR fallback for legacy thick clients.	Window titles, control labels, modal dialog text, tab order.
API (REST/GraphQL/SOAP)	Replays observed requests; introspects schemas where exposed; derives schemas from response shapes when not.	Endpoint signatures, response payloads, error codes, rate-limit headers.
Database (read-only)	Inspects schema, samples rows, follows foreign keys, profiles cardinalities.	Tables, columns, constraints, indexes, value distributions.
File drop / SFTP	Watches a directory; infers file format and record schema; builds a virtual stream from polled files.	CSV / fixed-width / EDI / XML payloads, naming conventions, arrival cadence.
Queue / event log	Tails the queue without acknowledging; reconstructs flows from message metadata and bodies.	Topic structure, message schemas, dead-letter contents, consumer-group lag.
Email / chat	Read-only mailbox or channel access; extracts approvals, escalations, and decisions that happen out-of-band.	Subject patterns, decision phrases, who responded to whom and when.
Spreadsheet	Treats the workbook as a database. Detects formulas, cross-sheet references, named ranges, conditional formatting as policy.	Formulas, structured ranges, validation lists, dashboard formulas.

WHAT "ANY" MEANS IN PRACTICE

Discovery does not require all surfaces to be present. A given source system typically exposes two or three of the surfaces above. Discovery picks the richest available surface for each region of the system — the database for entity shape, the UI for workflow flow, the email mailbox for the off-band approvals that the database doesn't see — and reconciles the evidence across surfaces during Capture.

The "any" claim is operational, not aspirational: if a human operator can do the work using the source system, Discovery can read enough to mirror it. If the work happens in a place no human ever sees — a black-box scoring

service that nobody at the customer can describe — Discovery will halt at that boundary and request either a black-box behavioral test set or human-supplied documentation. Chapter 7 quantifies how often this happens.

PROCESS TAXONOMY

Discovery is process-agnostic in the same operational sense. The Rule Extractor recognizes a small number of process shapes that recur across enterprise software:

- **Approval chains** — sequential or parallel approver routing with thresholds. Common in AP, expense, PO, hiring.
- **Routing decisions** — "this kind of thing goes to this queue" classifications. Common in support, claims, fraud, exception handling.
- **Reconciliation** — matching records from two streams against a tolerance. Common in bank-feed, AR, cash application.
- **Lifecycle state machines** — entities moving through named states with rules governing transitions. Common in tickets, deals, candidates, contracts.
- **Calculations** — derived values computed from inputs by rule. Common in pricing, commissions, tax, accruals.
- **Compositions** — assembling many small things into one larger thing under constraints. Common in order capture, BOM rollup, schedule builders.

Most enterprise workflows decompose into combinations of the above. Discovery's long-term memory tracks which shape combinations recur in which industries, and which extraction strategies converge fastest on each. This is what makes the second discovery run on a similar workflow significantly faster than the first.

ENGINE TAXONOMY

The "engine" Discovery captures is the rules layer underneath the workflow. Engines come in many forms:

- **Explicit BPMN / workflow engines** — rules expressed as diagrams or DSL. Easiest case: Discovery reads the diagram directly.
- **Configuration-driven systems** — rules expressed as data inside the source system (e.g. an "approval matrix" table). Discovery reads the configuration table and treats it as policy source.
- **Code-embedded rules** — rules expressed as conditionals in the source system's code. Discovery cannot read closed-source code; it infers rules from observed behavior on the surface.
- **Spreadsheet logic** — rules expressed as formulas in workbooks that the team has accidentally relied on. Discovery treats the workbook as a first-class engine and extracts the formulas as policies.
- **Tribal knowledge** — rules expressed nowhere, only known to humans. Discovery infers them from human decisions captured in the event stream — chats, emails, manual overrides — and prompts for human confirmation before promoting.

THE HARDEST CASE: TRIBAL KNOWLEDGE

The hardest engine to capture is the one that exists only in operators' heads. Discovery's strategy is to treat human decisions as the labeled training set: each manual override, each exception escalation, each "we always do X when Y" comment in chat is an example. The Rule Extractor proposes the rule that explains the observed decisions, tests it against held-out decisions, and promotes only the rules that survive the held-out test. The remainder is presented to humans as "*we observed these N decisions but cannot explain them automatically — please confirm or correct.*"

This is the gap between what Discovery does and what a process-mining tool does: the latter reports observed flows; Discovery *proposes the rule* that explains the flow and tests it. The output is policy, not a chart.

RULE CAPTURE METHODOLOGY

Rule capture is the chapter that determines whether Discovery is a useful tool or a flashy demo. The bar is high: the captured rules must be testable, replayable, and explainable; they must hold up when the source system shows them an example they have not seen before.

THE CAPTURE PIPELINE

1. **Event ingestion.** Surface Adapter normalizes the event stream into a typed event log.
2. **Decision detection.** The Rule Extractor identifies which events are *decisions* (state transitions, routing actions, approvals) versus which are *observations* (a record was viewed, a query was run).
3. **Hypothesis generation.** For each decision class, the extractor proposes candidate rules from a library of rule templates: threshold ("approve if amount < $X"), category ("route to queue Q if vendor type = T"), composite ("if A and B but not C, escalate").
4. **Held-out testing.** Each candidate is tested against held-out decisions of the same class. F1 is computed against the human-made decision as the ground truth.
5. **Disagreement triage.** Decisions where the candidate disagrees with the human are surfaced for review; many disagreements turn out to be the rule, not its violation.
6. **Promotion.** Candidates exceeding the F1 threshold are promoted to candidate Proto policies. Promotion to *active* policies is human-gated.
7. **Drift watch.** Active policies are re-tested against new decisions on a rolling window. Drift triggers re-capture rather than silent decay.

RULE TEMPLATES SUPPORTED IN V1.0

TEMPLATE	FORM	TYPICAL USE
Threshold	`action if numeric_field [op] value`	Auto-approval below a dollar amount.
Category	`action if categorical_field in {set}`	Route by vendor / region / class.
Composite	`action if conj/disj of conditions`	Multi-factor approval logic.
Lookup	`action via reference_table.match(field)`	Approval matrices, GL coding, tax lookups.
Tolerance	`action if abs(a − b) < tol`	Reconciliation, three-way match, variance.
State transition	`state[X → Y] if conditions`	Lifecycle automation.
Time-bound	`action if elapsed since t [op] interval`	SLAs, dunning, escalations.
Exception	`else action_E`	Catchall routes for unmatched cases.

WHAT WE CAPTURE VS. WHAT WE SURFACE

Rules with F1 above the configured threshold are captured automatically. Rules below threshold are surfaced as candidates with their failing examples; the human review step decides whether to broaden the rule, narrow it, or split it. Decisions that resist any rule template — genuinely judgement-call decisions — are surfaced as *policy gaps* and either become human-in-the-loop steps in the resulting Proto, or trigger a request for richer evidence (e.g. additional event volume).

The capture quality is bounded by the evidence quality. Discovery does not invent rules; it explains observed decisions. If the observed decisions are inconsistent across operators, the captured policy will reflect that inconsistency — and the disagreement report will say so explicitly. This is by design: hiding the inconsistency would let an inconsistent rule become silent policy.

EVALUATION FRAMEWORK

Discovery's job is to mirror — not to be plausible, not to look impressive in a screenshot, but to actually clone the behavior of the source system. The evaluation framework measures four things that, taken together, decide whether the mirror is real.

ILLUSTRATIVE — PRE-LAUNCH PROJECTION · ALL NUMERIC FIGURES IN THIS CHAPTER ARE PRE-LAUNCH PROJECTIONS. V1.1 WILL REPLACE THEM WITH MEASURED PRODUCTION DATA ONCE THE RESTRICTED PREVIEW COHORT COMPLETES INITIAL RUNS (TARGET: Q3 2026).

THE FOUR CORE METRICS

METRIC	WHAT IT MEASURES	HOW IT IS COMPUTED
Coverage	The fraction of the source system's observable workflow scopes that Discovery has mirrored.	Mirrored scopes / total inventoried scopes, weighted by observed event volume.
Fidelity	The agreement between captured policies and the source system's human decisions on held-out events.	F1 score of policy decision vs. human decision over a held-out window of recent events.
Replay accuracy	The agreement between Proto's end-to-end execution and the source system's end-to-end execution on the same fresh sample.	Fraction of replayed cases where Proto's final state matches the source system's final state on the same input.
Drift	The rate at which captured policies start disagreeing with the source system over time.	Rolling F1 over the last N days; drift event when F1 drops below the re-capture threshold.

DEFAULT SUCCESS CRITERIA

- **Coverage** ≥ 90% of high-volume workflow scopes (top quartile by event count).
- **Fidelity** ≥ 0.92 F1 on held-out decisions, per workflow scope.
- **Replay accuracy** ≥ 95% match on a sample of 200 fresh cases per scope.
- **Drift** alerts on F1 drop > 5 points over a rolling 14-day window.

These are the v1.0 defaults. Higher-stakes scopes (e.g. anything touching payments) ship with stricter defaults (98% replay, 0.95 fidelity); lower-stakes scopes can opt in to looser defaults if the operator's risk tolerance permits.

WHAT THE METRICS DO NOT MEASURE

The four core metrics measure agreement with the source system. They do *not* measure whether the source system is making the right decision. A high-fidelity mirror of a poorly-designed approval workflow is still a mirror of a poorly-designed approval workflow. Discovery's job is faithfulness; the human-review step is where the operator decides which captured policies to keep, which to revise, and which to retire as part of the migration.

SYNTHETIC BENCHMARK SUITE (ILLUSTRATIVE)

ILLUSTRATIVE — PRE-LAUNCH PROJECTION · PRE-LAUNCH PROJECTIONS. THE SYNTHETIC BENCHMARK SUITE IS A CONTROLLED TEST HARNESS; PRODUCTION RESULTS FROM THE V1.1 RELEASE WILL REPLACE THIS TABLE.

BENCHMARK	SCOPE	DECISIONS	COVERAGE	FIDELITY (F1)	REPLAY ACC.
SYN-AP-01	AP three-way match + tiered approval	12,400	96%	0.94	97%
SYN-AR-01	Cash application + dunning	8,200	93%	0.91	95%
SYN-PO-01	Multi-tier PO routing	5,100	98%	0.96	98%
SYN-CRM-01	Lead routing + qualification	22,500	94%	0.93	96%
SYN-HR-01	Hiring approval + offer routing	3,800	92%	0.90	94%
SYN-SUPP-01	Tier-1 ticket triage + escalation	31,200	97%	0.95	96%
SYN-EXP-01	Expense report approval + policy check	9,700	95%	0.92	95%
SYN-INV-01	Inventory replenishment triggers	4,500	91%	0.89	93%

ILLUSTRATIVE PRE-LAUNCH BENCHMARK TARGETS. EACH SUITE IS A SYNTHETIC SOURCE SYSTEM WITH KNOWN GROUND-TRUTH RULES; THE METRIC REPORTED IS DISCOVERY'S RECOVERY OF THE GROUND-TRUTH RULES FROM THE OBSERVABLE BEHAVIOR.

FAILURE MODES DISCOVERY IS DESIGNED TO SURFACE

FAILURE MODE	SYMPTOM	HOW DISCOVERY SURFACES IT
Ambiguous decision class	Same input, different decisions across operators.	Reports the operator-level disagreement and refuses to promote a policy until it is resolved.
Insufficient evidence	Decision class observed too few times to fit a stable rule.	Surfaces as a low-confidence candidate with the count; recommends a longer Observe window.
Black-box dependency	Source system delegates the decision to an opaque external service.	Halts the scope, reports the boundary, requests behavioral test set or documentation.
Drift during run	Source system's behavior changes mid-discovery (e.g. a config update).	Mid-run alert; offers to restart Capture for affected scope or freeze on pre-change behavior.
Schema gap	Field referenced in observed rules is not exposed by any surface.	Surfaces as a missing-field error with the rule that needs it; cannot promote.

READING THE TRACE

Every metric is reproducible from the trace. `GET /v1/discovery/:id/trace` returns the event log, hypothesized rules, held-out splits, and decision agreement — the audit substrate for any review.

CASE SKETCHES

Three anonymized walk-throughs from the restricted preview. Vendor names, customer identities, and absolute volumes are withheld; relative figures and the substantive findings are preserved.

ILLUSTRATIVE — PRE-LAUNCH PROJECTION · PRE-LAUNCH PROJECTIONS DRAWN FROM PREVIEW-COHORT ENGAGEMENTS; PRODUCTION RESULTS IN V1.1 WILL REPLACE THESE SKETCHES WITH ATTRIBUTABLE CASE STUDIES ON CUSTOMER PERMISSION.

CASE A — A MAJOR LEGACY ERP, AP MODULE

Source: a Tier-1 legacy ERP running on-prem at a multinational manufacturer; AP module covering ~120k vendor invoices/month across three currencies and twelve approval tiers.

Surface: read-only Oracle DB connection (the legacy ERP's underlying store), screen-scrape of the AP approver UI for in-flight items, mailbox access for the approver-replies channel where ~30% of high-value approvals were happening out-of-band.

Discovery scope: three-way match + tiered approval routing.

Outcome (illustrative):

METRIC	RESULT
Inventoried entity types	47
Mirrored entity types	44 (94%)
Hypothesized rules	312
Promoted to candidate policies	281 (90%)
Promoted to active (after human review)	256 (82%)
Replay accuracy on 500 held-out invoices	96.4%
Wall-clock to first replay	~9 days

The interesting finding: the email-mailbox surface was load-bearing. The "official" ERP workflow had a single approval step; the actual workflow had a parallel email-based escalation path operators used for time-sensitive cases. Discovery captured that side-channel and modeled it as an explicit dual-route policy. The mirror would have been structurally wrong without it — and no documentation captured it.

What halted: three entity types depending on an opaque tax-determination service were left out of the mirror; Discovery surfaced the boundary and the customer chose to wrap the tax service as a Proto-callable dependency rather than mirror it.

CASE B — A TIER-1 SAAS CRM, LEAD ROUTING

Source: a major SaaS CRM at a SaaS company with ~40k inbound leads per quarter; lead routing rules had accumulated over five years across three round-robin algorithms, two custom Apex-class equivalents, and a number of managers' private "always assign these to me" overrides expressed as Slack rules.

Surface: CRM REST API, audit-log API, Slack mailbox for the routing channel, sales-ops admin UI for the configuration surface.

Discovery scope: lead routing + initial qualification.

Outcome (illustrative):

METRIC	RESULT
Hypothesized routing rules	73
Promoted to candidate	66 (90%)
Promoted to active	52 (71%)
Surfaced inconsistencies (operator-level disagreement)	14
Replay accuracy on 1,000 held-out leads	93.1%

The interesting finding: Discovery flagged 14 routing decisions as inconsistent across managers — the same lead profile routed to different reps depending on who was paying attention. Suspected for years, never quantified. The remediation was a sales-ops decision about which manager's rule was authoritative; Discovery's output became the input to that decision.

What halted: a recently-introduced "AI lead scoring" service was treated as a black box; Discovery captured the input-output behavior but kept the service as a Proto-callable dependency rather than claiming it had recovered the rule.

CASE C — SPREADSHEET-DRIVEN APPROVAL PIPELINE

Source: a department's shared Google Sheet acting as the de-facto approval system for marketing-spend requests at a mid-market retailer; ~600 requests/month, 4 approval tiers, formulas and named ranges encoding the rules.

Surface: Sheets API (read-only), Slack mailbox for approver pings, approvers' calendars (to detect when approvals weren't happening because someone was in a meeting). **Scope:** the entire spend-approval pipeline.

Outcome (illustrative):

METRIC	RESULT
Workbook formulas captured	87
Promoted to active policies	81 (93%)
Replay accuracy on 200 held-out requests	97.0%
Wall-clock to first replay	~2 days

The interesting finding: cell J42 had a three-year-old comment explaining a rule change the team had forgotten. Discovery flagged it; the captured policy was corrected before promotion.

HANDOFF TO PROTO

Discovery is finished when its output runs as a live Proto. The handoff is a contract: a typed packet describing entities, policies, and topology that Proto knows how to execute.

THE HANDOFF CONTRACT

CAPTURE	BECOMES IN PROTO	VERIFIED BY
Mirrored entities	Working data + state model	Schema validation against ERP.AI core data model.
Captured rules	Proto policies	Policy linter; held-out replay before activation.
Observed flows	Proto topology (sub-agent graph)	Topology checker; loop-bound and DAG checks.
Discovered surfaces	Proto tool registrations	Sandbox call test against each registered tool.
Trace	Initial knowledge graph for the new Proto	Provenance check; every imported memory carries DISC-id.

API SURFACE FOR THE HANDOFF

```
POST /v1/discovery/:id/promote
{
  "promotion_scope": "all",          // or list of policy_ids
  "target": { "proto": "auto-create" },
  "human_approver": "user_email",
  "activate": false                  // policies become "active" only after explicit activate call
}

# Returns
{
  "proto_id": "PRT-9012",
  "policies_promoted": 256,
  "entities_imported": 44,
  "topology_nodes": 18,
  "trace_url": "https://app.erpai.com/trace/PRT-9012",
  "activation_required": true
}
```

PARALLEL RUN, NOT CUTOVER

The default handoff puts the resulting Proto in *parallel-run* mode: Proto receives the same inputs as the source system, executes its own decisions, and writes them to a parallel-trace store rather than to the source system or to production ERP.AI data. Operators compare the parallel decisions against the source system's decisions over a

configured window. Activation — switching the Proto from parallel-run to authoritative — is an explicit, separately-authorized step.

This is the operationally conservative posture. It permits faster iteration than a traditional cutover (because the target is already running), preserves rollback (because the source is still authoritative), and surfaces drift (because the parallel-trace store accumulates the disagreement set). Most preview-cohort engagements have run in parallel for two-to-six weeks before activation per scope.

WHAT PROTO DOES THAT DISCOVERY DOES NOT

Discovery's job ends at handoff. Proto's job begins at handoff. Proto extends the captured behavior with the full Proto capability set: runtime tool synthesis for new integrations, multi-agent topology for parallel work, the ORAI loop for in-mission iteration, and the long-term memory that lets Proto improve on the captured baseline rather than just replicate it. The full Proto surface is documented at /proto and /proto.md.

GUARDRAILS, SAFETY, AND HUMAN GATES

Discovery operates against production systems on customer property. The safety envelope reflects that. Five primitives, each enforced at the platform level rather than at the agent's discretion.

1 · READ-ONLY BY DEFAULT

Discovery's default credential profile is read-only. This is not a soft default the agent can override at runtime; it is a property of the credential the platform issues to the discovery sandbox. Write-back to the source system — which Discovery will rarely need, but may need for parallel-run synchronization — requires a separate, scoped, time-boxed write credential issued only after an explicit human-approved policy is attached. There is no path by which Discovery can write to a source system without a human having taken the explicit step.

2 · TOKEN BUDGET

Every Discovery run is bounded by a token budget configured at start. The Coordinator emits a budget-warning event at 75% consumption and halts at 100%. There is no implicit extension. Operators extend explicitly, with the new ceiling and a reason captured in the trace. The default budgets in Chapter 4 are sized for typical scopes; outliers (unusually rich source systems, very long observation windows) require explicit larger budgets at run start.

3 · HUMAN-APPROVAL GATES

The configurable `human_approval` array controls which actions block on a human signal. The defaults for v1.0 are conservative:

- **mirror_create** — first-time allocation of a mirrored entity type
- **rule_promote** — promotion of a candidate policy from "candidate" to "active"
- **scope_expand** — expansion of discovery scope beyond the originally-configured workflow boundary
- **write_credential_attach** — attaching a write credential to the run
- **handoff_promote** — promotion of the resulting Proto out of parallel-run into authoritative mode

Operators may add gates (e.g. on every entity type, not just the first); operators may not remove the platform-mandatory gates (write credentials, handoff promotion).

4 · SANDBOXED EXECUTION

Each discovery run executes in an isolated sandbox keyed to the discovery_id. Sandboxes share no state with each other. Cancellation rolls back the sandbox; the only artifact that persists is the trace, retained per the configured retention policy.

5 · FULL REPLAYABLE TRACE

Every observation, every hypothesized rule, every promotion, every Mirror Allocator decision is written to the trace. The trace is sufficient to reconstruct what Discovery saw and decided, and can be replayed against a captured event stream to reproduce the run deterministically. The trace is the audit substrate for compliance and for internal review.

DATA HANDLING

CLASS	DEFAULT TREATMENT
Source-system credentials	Stored in customer-controlled vault; never inlined in trace; rotated per policy.
Source-system data (entities, events)	Held in the discovery sandbox; not exfiltrated to ERP.AI core stores until handoff promotion is approved.
Personally identifiable information	Detected and masked in the trace; original values remain in the sandbox under scoped access.
Captured policies	Versioned, signed by the Discovery run that produced them, attributable to a source event.
Long-term memory	Tenant-scoped by default; cross-tenant pattern reuse opt-in.

COMPLIANCE POSTURE

Discovery inherits the ERP.AI platform compliance posture (see /enterprise for the full enterprise-grade controls). Notable Discovery-specific items:

- **Right to explanation.** Every captured policy carries a provenance pointer to the source events that justified it. An auditor can answer "why does Proto make this decision" by reading the trace.
- **Right to deletion.** Cancellation of a discovery run wipes the sandbox; cancellation of a promoted Proto reverts the imported policies and entities, with a separate workflow for the long-term-memory delta.
- **Cross-border.** Sandboxes can be pinned to specific regions; the trace inherits the sandbox region.
- **Separation of duties.** The operator who configures a discovery run cannot be the operator who approves the handoff; this is enforced at the platform level.

WHAT DISCOVERY REFUSES TO DO

- **It will not promote a rule it cannot explain.** Rules without a human-readable provenance are not promoted regardless of fidelity score.
- **It will not silently broaden scope.** Scope expansion is human-gated by default.
- **It will not write to a source system without a separate write credential.** The credential model is enforced at issuance, not at the agent.
- **It will not delete from a source system, ever.** No discovery-issued credential includes delete permission. Period.
- **It will not bypass a failing held-out test.** A candidate that does not meet the fidelity threshold is surfaced, not promoted — even if the operator asks it to.

INCIDENT RESPONSE

Discovery emits structured incident events on any halt, gate failure, or budget breach. Incidents are visible in the run trace and via the standard ERP.AI alerting channels (web hook, email, on-call rotation). The trace identifies the precise step at which the incident occurred and the configured remediation policy.

GLOSSARY

TERM	DEFINITION
Discovery	A Proto-class agent that mirrors a source system into ERP.AI by discovering, observing, and capturing its workflows and rules.
Discovery run	A single invocation of the agent against a configured source system, scope, depth, and budget; identified by a DISC-id.
Surface	A category of evidence the source system exposes (UI, API, DB, file, queue, mailbox, spreadsheet); see Chapter 5.
Mirror	The set of ERP.AI-side entities, fields, and states that Discovery has allocated to represent the source system's structure.
Capture	The set of policies Discovery has extracted from observed source-system decisions.
Policy	A testable, replayable rule expressed in Proto's policy DSL; produced by Discovery, executed by Proto.
Held-out	A subset of observed events withheld from rule generation, used to validate hypothesized rules.
Fidelity	Agreement between captured-policy decisions and source-system human decisions on held-out events; measured as F1.
Replay accuracy	Agreement between Proto's end-to-end execution and the source system's end-to-end execution on a fresh sample.
Drift	Decay in policy agreement over time, typically caused by changes in the source system or in operator behavior.
Parallel run	A handoff mode in which the resulting Proto receives the same inputs as the source system but writes only to a parallel-trace store.
Activation	An explicit, separately-authorized step that switches a parallel-run Proto to authoritative.
Trace	The full replayable log of a Discovery run; the audit substrate for compliance and review.
Long-term memory	Cross-run knowledge held by the Discovery agent; tenant-scoped by default.
Tribal knowledge	Rules expressed nowhere in the source system, only in operators' heads; Discovery infers them from observed human decisions.

REFERENCES & FURTHER READING

- **/proto** — Proto, the autonomous agentic workflow engine that executes Discovery's output.
- **/proto/discovery** — Discovery's product page and live documentation.

- **/proto/discovery.md** — Agent-readable API surface for Discovery.
- **/proto.md** — Agent-readable API surface for Proto.
- **/enterprise** — ERP.AI enterprise compliance, certifications, and deployment posture.
- **/changelog** — Versioning policy and release notes; v1.1 of this whitepaper will replace illustrative figures with measured production data.

DOCUMENT

DISC-WP-001 · v1.0 · April 2026. Issued by ERP.AI. All quantitative figures herein labeled *illustrative* are pre-launch projections subject to replacement in v1.1 by measured production data. Anonymized case sketches preserve relative figures and substantive findings; vendor names, customer identities, and absolute volumes are withheld until customer consent permits attribution.