The Autonomous Research & Invention Stack

A team of agents that researches almost any question, weighs the evidence, and returns a buildable proposal, sharper the more it runs.

INTRODUCTION

What this is, and why it exists

This document describes an autonomous system that takes a question from start to finish and returns a proposal solid enough to act on. It is one engine within TARS, the multi-agent platform CASE runs its own ventures on; the pages that follow open it up, step by step, in plain language.

The hard part of research was never finding an answer. It was being able to trust one enough to build on it.

The problem

A capable language model can answer almost any question, but a single prompt returns a single pass: an answer that is often shallow, hard to verify, and gone the moment the session closes. For an operator making real decisions across many fronts at once, that is not a foundation to build on. What was missing was never a better model; it was a system built around one, able to research in depth, test its own conclusions, and keep what it learns.

What it is

This stack is that system. A team of specialised agents researches a question in rounds, not a single pass: it gathers widely, tests every finding for and against trusted sources, and admits only what holds up. When a finding won't hold or a trialled idea fails, it loops back and researches deeper, round after round, until it genuinely understands — then turns that into a concrete, buildable plan. No finding is trusted on a single source. What comes back is research an operator can act on directly, not a draft to recheck by hand.

What it enables

Because the work is deep, self-checking, and retained, the same engine can be aimed at almost anything — from a huge, open-ended ambition it breaks into many rounds of research down to a single focused question — and trusted to return something usable. It extends itself to the task, standing up the tools a new problem needs. And because every result is kept, the system compounds: each question answered sharpens the next. One operator gains the reach of a research team that does not forget.

Where it fits in TARS

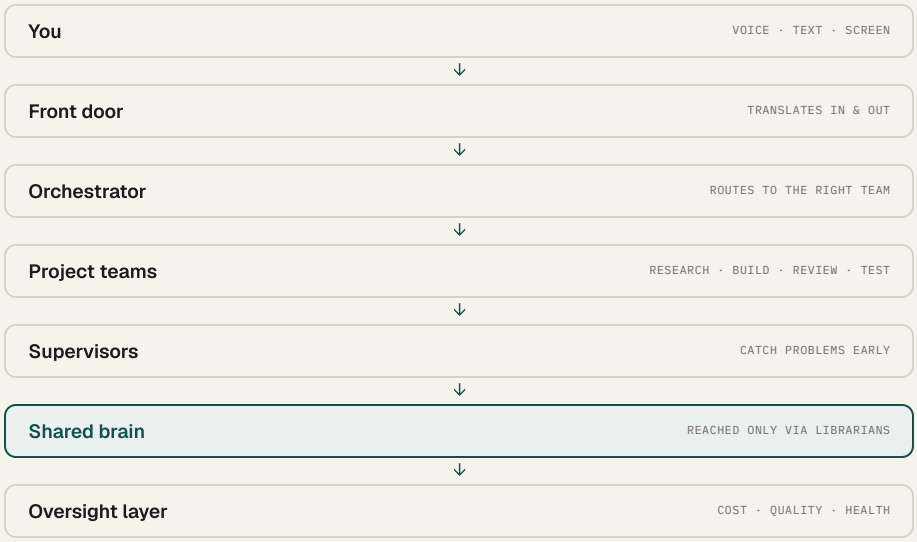
The engine does not work alone. It is one part of **TARS**, the multi-agent operating system CASE runs its own ventures on: a layered structure of agents organised around a shared memory, operating without pause. Within TARS it is **general-purpose infrastructure**, fired three ways: by an operator on demand; by a project team mid-build, which gets the research back, reviews it, then builds; and in idle hours, when the system turns the engine inward to research its own gaps and surface proposals by morning. Whoever fires it, it delivers understanding and a plan and **hands off** — it does not build or deploy the end system itself. This document describes the engine in general form, independent of any single application.

The pages ahead

HOW IT FITS · TARS IN PRODUCTION

Where this lives

On its own, this is a research engine. Inside **TARS** it is **general-purpose infrastructure**: any team can aim it, on demand, at whatever needs researching — a venture to weigh, a rough idea to turn into a plan, a client's market to map, a technical decision to settle. The overnight loop that researches TARS's own gaps is one caller among those, not the whole story.



ONE CALLER, AMONG OTHERS

The overnight loop

Any team books this engine when it needs something researched. Among those callers, **dream agents** run it in idle hours against the shared brain: they look for what's vague, half-finished, or unanswered, research it, and turn it into a proposal.

Every read and write to the brain goes through a **librarian**. No agent touches the memory directly.

In the morning the proposals are waiting for the owner to accept, change, or discard. The system suggests; the person decides.

↻ each night it knows a little more

Figure. The engine is reached through librarian agents over the shared brain, under an oversight layer watching cost and quality. The same loop that researches the outside world can be turned inward to research TARS itself, but its only output, on any target, is a proposal that surfaces to the owner.

What this page adds

The rest of this document explains the research loop as a thing in itself. This page says where it sits. TARS is a layered system: a person at the top, a chain of agents that route and do the work, a shared memory underneath, and an oversight layer watching the whole thing. The research loop isn't a separate product bolted on; it's general-purpose infrastructure any team can aim, on demand, at whatever needs researching, from a venture to weigh to a client's market to map; a project team even fires it mid-build and builds from what comes back. Turned inward overnight is one of its jobs, not the whole of it.

The brain, reached through librarians

The shared memory is the most valuable and most sensitive part of the system, so nothing reaches it directly. A class of **librarian** agents handles every lookup and every write on behalf of the others. That keeps the memory consistent, keeps a record of who changed what, and means the rest of the agents never need to know how the memory is built.

It only ever proposes

Whatever the engine is pointed at, it can research, plan, draft, and stand up throwaway experiments to test its own ideas, but it cannot build or deploy the end system on its own. Whatever it produces arrives as a **proposal** for a person to weigh and build. This is the firm line: the system gets to think freely precisely because acting still needs a person.

Why it compounds

Because each round's work is banked into the same shared memory the next round reads from, the system doesn't just answer questions. It accumulates. The longer it runs, the more it already knows, and the sharper its proposals get. That compounding is the point of building it this way rather than as a tool you call when you need it.

FIGURE 1 · THE WHOLE STACK AT A GLANCE

It builds knowledge, compounds it, then invents

Three movements, run as a loop. A round gathers and hardens knowledge — looping back to research deeper whenever a finding won't hold; the round repeats and the knowledge compounds; and when a task needs a new design, an invention layer turns the knowledge into a buildable solution. The same engine is fired at anything from a huge ambition to a focused question, every stage run by specialist agents.

1

ONE ROUND

Gather wide, then harden what holds

A round is one pass at a question. An `interviewer` frames it; `five scrapers` cast a wide net across code, papers, write-ups, forums and rival systems; a `checker panel` tests each finding for and against trusted sources and scores its confidence; and specialist reviewers loop the weak spots back for more research until what remains is solid.

2

MANY ROUNDS

The knowledge compounds, round over round

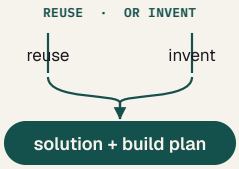
The round runs again and again, and each new round starts from everything the earlier ones confirmed, plus what the stack learned about how to research the topic. The knowledge base grows, later rounds get faster and sharper, and the gaps keep shrinking. The stack grows more capable the longer it runs.

3

THE INVENTION LAYER

Turn the knowledge into a buildable solution

A `pattern-mapper` finds reusable structures; to invent, the layer recombines them into new candidate systems and a `voting panel` keeps the strongest. It reuses a proven approach when one fits and invents only when none does, producing a solution and a build plan, handed over as a proposal.



Running throughout: safety checks · an effort budget · save-and-resume across machines · a paywalled-source fetcher · the search tooling, and the round pauses if anything looks wrong.

ROUND SETUP · INTAKE

What are we being asked?

Think of it as the site visit before the quote. Before any searching starts, a short interview pins down the real question and decides how deep this round should go, so effort is scoped to the task instead of guessed at.

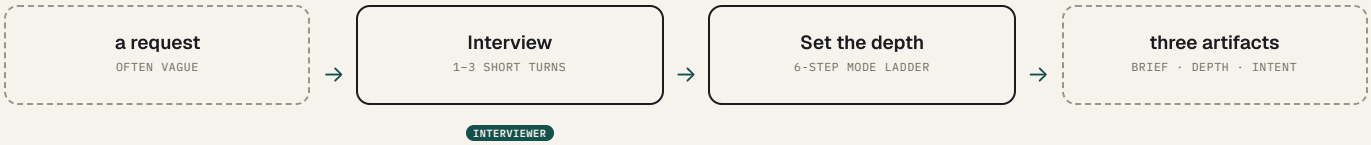


Figure (intake). A request enters; an interviewer agent clarifies it over one to three turns, sets the round's depth, and emits three hand-off artifacts that everything downstream reads. If a request already arrives fully specified, the interview is skipped.

What this step does

Most requests arrive underspecified: "research X" without saying how broad, how deep, or what "done" looks like. Intake turns that into a precise, scoped brief, and decides the round's **depth** up front, so a quick lookup doesn't get a week-long treatment and a hard problem doesn't get a shallow one.

What's inside it

The interview — 1–3 turns

A structured back-and-forth that draws out scope, the success criteria, and any hard constraints. It asks only what it needs, and stops as soon as the question is clear.

The depth ladder — 6 checks

A fixed priority order that picks one of four depths (*quick, medium, deep,* or *heavy-invention*) from explicit hints first, then time and effort budgets, falling back to a sensible default and an escape hatch.

Three artifacts — the hand-off

A *depth signal* (how hard to go), a *refined prompt* (the full question every later stage works from), and an *intent record* (a structured summary of what's wanted and why).

How data moves

In: a raw request, in whatever shape it arrives. **Out:** the three artifacts above. Nothing downstream ever sees the original vague request. It only sees the refined prompt and the depth setting, which keeps every later stage working from the same clear, agreed scope.

ROUND SETUP · PLAN THE SEARCH

Planning the search

Think of it as a trip planner breaking one journey into legs. Before any scraping starts, the round breaks the question into a concrete plan: which topics to chase, how wide to go on each, and how much of the round's effort each one earns.

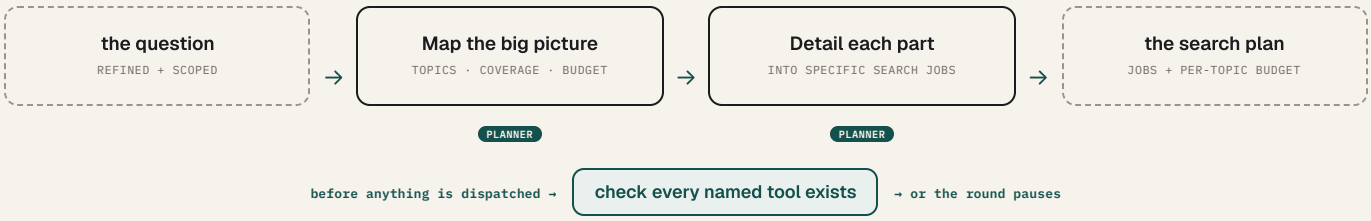


Figure (plan). Planning happens in two passes: a wide one that lays out the topics and splits the effort budget, then a close one that turns each topic into specific search jobs. A final check confirms every tool the plan names is actually available, so a round never sets off referencing something that isn't there.

What this step does

A research question is too big to hand to a scraper as-is. This step turns it into a layered plan. The first pass works at altitude: it names the **topics** worth investigating, sketches how much **coverage** each needs, and divides the round's effort budget between the later stages. The second pass drops down a level and turns each topic into the actual **search jobs**: what to look for, and which kind of source to look in.

What's inside it

The wide pass `map`

Reads the scoped question and lays out the topics, the rough depth each one deserves, and how the effort budget is split across gathering, checking and inventing.

The close pass `elaborate`

Takes each topic and writes the concrete search jobs for it, phrased so the right kind of source (papers, code, write-ups, forums) gets targeted, not just a generic query.

Tool-existence check `guard`

Every scraper, helper and skill the plan names is confirmed to exist before dispatch. A plan that references a missing tool pauses the round instead of failing halfway through.

How data moves

In: the refined question and the project scope handed over by intake. **Out:** a single search plan: the list of topics, the specific jobs under each, and the per-stage effort budget. Every later stage reads this one plan, so the whole round stays pointed at the same agreed set of questions.

ROUND SETUP · BACKGROUND BRIEF

The background brief

Think of it as a briefing pack dropped on every desk before the meeting. A lightweight librarian reads what the knowledge base already holds on the topic and writes a short, size-controlled brief, so the heavier stages begin oriented instead of from a blank page.

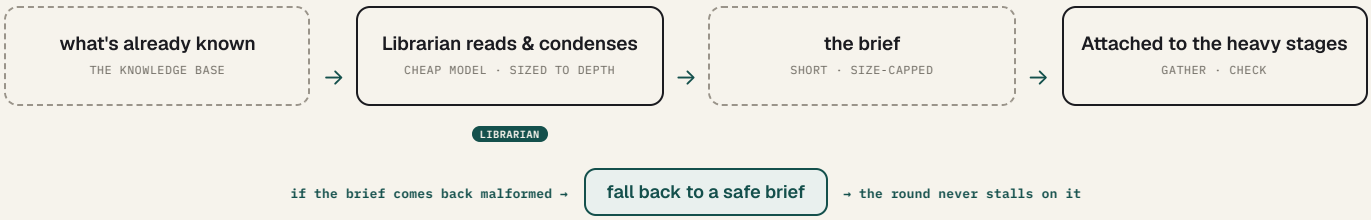


Figure (brief). A cheap model pulls the relevant prior knowledge and condenses it to a brief sized to the round's depth. The brief rides along with the wider, costlier stages; the lighter stages skip it. If the brief is ever malformed, a guaranteed-valid fallback is used so a bad brief can't block the round.

What this step does

Every round can stand on what earlier rounds confirmed. Rather than make each expensive agent re-discover that context, a small, cheap model acts as a **librarian**: it retrieves the knowledge relevant to this topic and condenses it into a brief. The brief is then handed to the stages that gain the most from context, so they open already knowing what's been settled and what's still open.

What's inside it

The librarian — `cheap model`

A low-cost model does the retrieval-and-summarise job, not the hard reasoning. Keeping it cheap means every round can afford a fresh brief without taxing the budget.

Size budget — `per-depth caps`

The brief's length is capped by the round's depth: a quick lookup gets a few lines, a deep round a fuller picture. Context is matched to the effort the round warrants.

Safe fallback — `graceful degrade`

If the librarian returns something malformed, a guaranteed-valid fallback brief is substituted. The round proceeds on a known-good brief rather than choking on a broken one.

How data moves

In: the round's topic and depth, plus the confirmed knowledge base. **Out:** a short brief, prepended to the stages that benefit most, the wide, expensive ones. The cheaper stages skip the brief entirely, so context is spent only where it changes the outcome.

ONE ROUND · GATHER

Casting a wide net

Think of it as casting a wide net — five boats fishing different waters, one of them a skeptic. Five scrapers search at once, each aimed at a different kind of source, so the round surfaces far more than any single search would, including the contrarian takes the friendlier sources tend to skip.

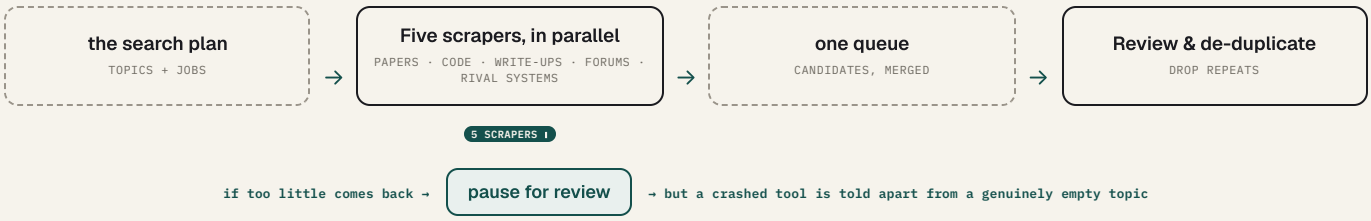


Figure A. Five scrapers run in parallel, each pointed at a different kind of source. Their results merge into one de-duplicated queue of candidate findings. If the haul is too thin the round pauses, but only after checking whether a tool simply failed, so it never pauses for the wrong reason.

What this step does

Gathering is meant to be broad, not precise. Five scrapers run at the same time, each aimed at a different kind of source, so a finding that appears in only one corner of the web still gets seen. The point is recall: surface everything that might matter now, and let the next stage decide what's real. One of the five is deliberately a **contrarian**, hunting for the critical and alternative takes that marketing-friendly sources quietly leave out.

What's inside it

Five parallel scrapers — `recall first`

Academic papers, open-source code, practitioner write-ups, and community forums: four angles on the same question, run at once so no single source's blind spot decides the round.

The contrarian fifth — `counterweight`

A scraper whose only job is to find the dissenting and alternative views, a built-in counterweight to the optimism the other four tend to inherit from their sources.

Thin-return + tool checks — `guard`

Too little back pauses the round, but a crashed scraper is told apart from a genuinely empty topic first, so a tool failure doesn't get mistaken for "nothing exists."

How data moves

In: the search plan. **Out:** one merged, de-duplicated queue of candidate findings, wide but entirely unproven. Everything here is a candidate only; nothing is allowed to count until the next stage has checked it.

ONE ROUND · VERIFICATION

Does it hold up?

Think of it as a courtroom, where every claim is put on trial. Every candidate finding from the wide search is tested against trusted, independent sources before it's allowed to count. What survives comes out marked true, false, or unsure, each with a confidence score.

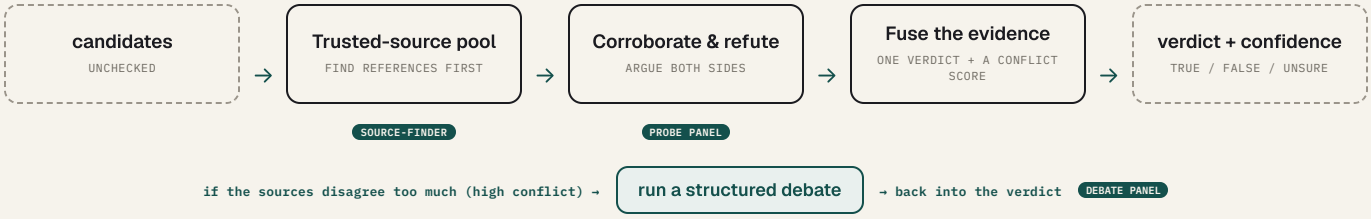


Figure B. Verification runs left to right. Solid boxes are work done by agents; dashed boxes are the data that moves between them. Most findings settle at the fuse step; only genuinely conflicting evidence is escalated to the debate.

What this step does

The wide search casts a deliberately broad net, so most of what it surfaces is unproven: promising-looking claims, forum opinions, vendor blog posts. Verification is the gate that decides what's real. It never trusts a finding on its own; it checks each one against sources it has reason to trust, and it keeps a record of *how* sure it is, not just yes-or-no.

This is also where **community signal earns its place**: a pattern that keeps coming up on forums is only promoted if it also lines up with a reputable source; otherwise it's held as "needs corroboration."

What's inside it

Trusted-source pool — references first

Before judging anything, it first assembles a set of reputable references for the topic, so every check is made against quality sources, not whatever the open web returned.

Corroborate & refute — argue both sides

Rather than asking "is this true?" once, a panel of probe-agents argues both sides: some try to confirm the claim, others try to break it. Claims that only look right at a glance fail here.

Fuse the evidence — fuse the signals

The for-and-against signals are combined into one confidence number, and the method also measures how much the sources *disagree*, a single conflict score.

Debate on conflict — on a clash

When that conflict score is high, agents each speaking for a different source debate across a few rounds until they converge on one reading, instead of the system silently averaging away a real disagreement.

How data moves

In: the batch of unchecked candidate findings from the wide search. **Out:** the same findings, each now labelled true, false, or unsure with a confidence score, plus one round-level number, the *corroboration ratio* (how much of the batch held up). If that ratio drops too low, a safety check pauses the round for review rather than letting a weak batch flow downstream.

ONE ROUND · VALIDATION

Promote, refine, or hold

Think of it as a customs gate at the border of the permanent record. The checked findings are judged once more as a set. Each one is either promoted into the confirmed knowledge, sent back to be strengthened, or held as a proposal until a later round can corroborate it.

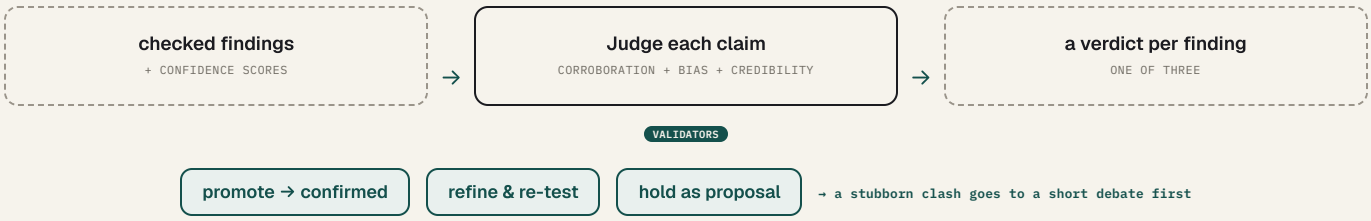


Figure C. Each checked finding is judged across the whole batch and lands on one of three outcomes. A finding needs independent support to be promoted; a partly-supported one is refined and re-tested; an interesting-but-unproven one is held for a later round. A conflict that won't settle is sent to a short debate before it can force a pause.

What this step does

The previous stage checks each finding against trusted sources. Validation decides what that verdict *means* for the knowledge base. Looking across the whole batch, it sorts every finding into one of three fates: confirmed and promoted, promising-but-partial and sent back to be strengthened, or unproven-but-worth-keeping and parked as a proposal. It's also where the credibility and bias checks live that stop thin evidence from being counted as strong.

What's inside it

Corroboration check — ≥2 sources

A finding needs independent support before it can be promoted. A single source, however confident, is held, not confirmed.

Bias & credibility audit — weigh, don't delete

A source pitching its own product is kept out of the "independent" count but not thrown away; a named expert outweighs an anonymous claim.

Three-way verdict — promote · refine · hold

When a finding is unevenly supported, it isn't all-or-nothing: the well-backed part is promoted and the thin part is broadened until the claim matches its evidence.

Debate on a clash — before a pause

A contradiction that resists the normal checks goes to a short structured debate. Only if that still can't resolve it does the round pause for a human.

How data moves

In: the checked findings with their confidence scores. **Out:** the promoted findings (into the confirmed knowledge), the refined ones (back for another pass), and the held ones (parked as proposals). Contradictions that survive several passes pause the round for review rather than being silently resolved.

ONE ROUND · GAP-FILL

Closing the gaps

Think of it as the final walk-through, snagging list in hand. Where validation finds holes, this stage sorts each gap by type and sends it back to exactly the right kind of search, looping with the validator a few times until the gaps close or it runs out of tries.

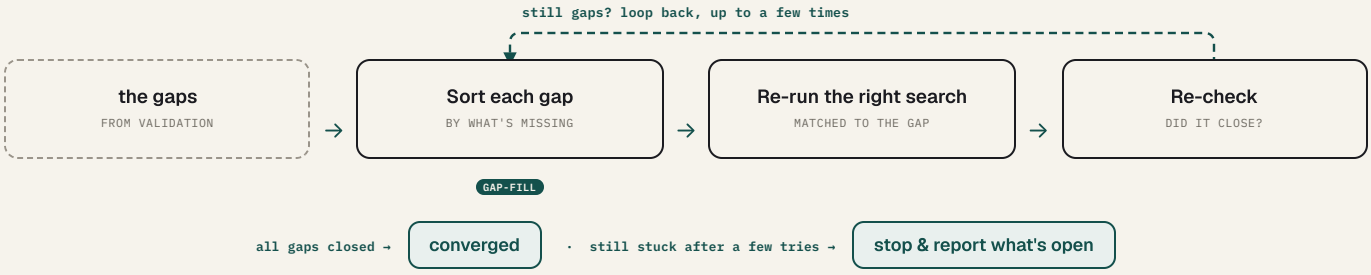


Figure D. Each gap is sorted by what it's missing and routed back to the stage that can close it; the validator then re-checks. The two loop until no gaps remain, or until a few passes go by without progress, at which point the round stops and reports what's still open rather than looping forever.

What this step does

Validation hands back what didn't fully hold: claims with only one source, partly-corroborated ones, and conflicts it couldn't settle. Each kind of hole needs a different fix, so this stage sorts the gaps by type and routes each to the matching action: a fresh wide search for the lonely claims, a re-check for the partial ones, a debate re-run for the conflicts. Then the validator looks again, and the two stages loop, closing gaps a pass at a time.

What's inside it

Sort by gap type `triage`

Every gap is labelled (single-source, partly-corroborated, or unresolved-conflict) because the right fix is different for each.

Route to the matching fix `target`

Each type is sent back to the stage that can actually close it: more searching, more checking, or another round of debate. No blanket re-do.

Close-the-loop check `capped`

The loop ends when no gaps remain, and is capped at a few passes, so a stubborn gap is reported as open rather than churning the round indefinitely.

How data moves

In: the gaps the validator surfaced. **Out:** closed gaps folded back into the findings, and any that couldn't be closed handed forward to the next round to try with fresh context. Nothing is quietly dropped. An unclosed gap is recorded and carried on, not forgotten.

THE INVENTION LAYER

From knowledge to a buildable solution

Think of it as an inventor's workshop: reuse a proven part before you fabricate a new one. With the knowledge hardened, the invention layer looks for a proven approach that fits the problem, and when none does, recombines what's known into new candidate designs, keeps the strongest, and hands over a proposal.

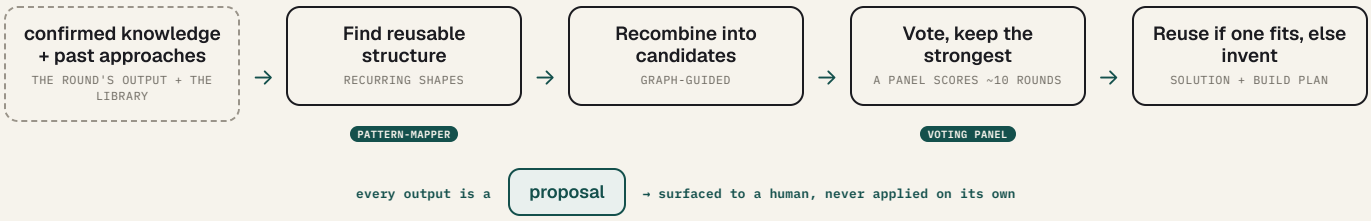


Figure E. The layer finds reusable structure, recombines it into candidate designs, and votes to keep the strongest. It prefers a proven approach when one already fits, and packages the answer as a build plan. Everything it produces is a proposal for a human to weigh, not an action.

What this step does

This is where research becomes a buildable answer. The layer reads everything the rounds have confirmed, plus the library of approaches earlier rounds already validated. It looks for reusable **structure** (the shapes and relationships that recur across findings) and recombines them into candidate designs for the problem at hand. A panel votes, and only the strongest survive. The intended posture is **reuse-first**: if a proven approach already fits, return that; invent a new combination only when nothing does.

What's inside it

Find reusable structure `map`
Pulls the recurring shapes and relationships out of the confirmed findings, the raw material a new design can be built from.

Recombine into candidates `graph-guided`
Combines those structures into candidate designs, steered by the existing links in the knowledge graph so the combinations are grounded rather than random.

Vote, keep the strongest `ensemble`
A panel scores roughly ten rounds of candidates and the majority view wins: a wide search for ideas, narrowed to the few worth proposing.

Reuse-or-invent + build plan `the output`
Return a proven approach when one fits instead of discarding it as too familiar, and package the answer as a concrete solution plus steps to build it.

How data moves

In: the confirmed knowledge and the accumulated library of validated approaches. **Out:** a small set of proposals, each carrying the evidence that supports it, handed to a human as suggestions to weigh, never as decisions already taken.

END OF ROUND · COMPOUNDING

Banking what was learned

Think of it as a logbook closed at the end of each round. When a round ends, its confirmed findings are written into a growing library: the facts, the approaches that worked, and what's still uncovered, so the next round doesn't start from zero. This is what makes the system sharpen the longer it runs.

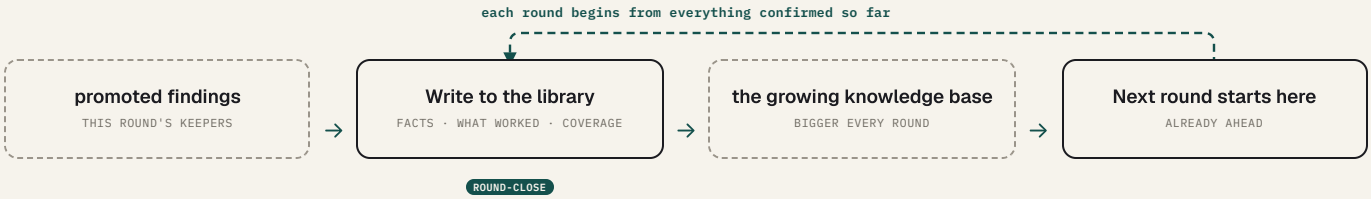


Figure (round-close). Every confirmed finding is banked three ways: as a fact, as a reusable approach, and as an update to the coverage map. The next round reads all three on start-up, so it begins already knowing the territory. The longer the system runs, the further ahead each round starts.

What this step does

A round's value shouldn't evaporate when it ends. At close, every confirmed finding is banked three ways: as a **fact** in the confirmed knowledge, as a reusable **approach** in a skills library, and as an update to the **map** of what's been covered and what's still open. The next round reads all of this when it starts, so it begins already knowing the territory, and the invention layer can reach for approaches that proved out in earlier rounds.

What's inside it

Skills library `reusable`
Each confirmed approach is saved as a compact, reusable entry, the store the invention layer pulls from when it looks for something proven that already fits.

Trajectory log `the trail`
A lightweight record of which findings held up, were later superseded, or sparked an invention, so it's possible to trace how the knowledge actually evolved.

Coverage map `aim the next round`
What's been answered and what's still open, updated each round, so later rounds point straight at the gaps instead of re-treading settled ground.

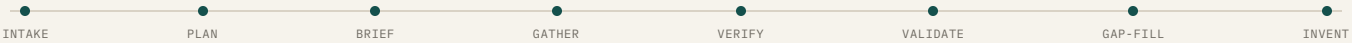
How data moves

In: the round's promoted findings. **Out:** three durable artifacts: the skills library, the trajectory log, and the coverage map, all read at the start of the next round. Nothing that's already been settled gets re-derived; each round inherits the last one's work.

ACROSS EVERY STAGE · THE SUBSTRATE

Running throughout

Think of it as the wiring and circuit-breakers behind the walls. Four systems run underneath every stage of every round: safety checks that can pause it, an effort budget that keeps it from overrunning, continuous save-and-resume so a round survives a crash or a move to another machine, and a sealed boundary that keeps anything pulled from the open web from reaching the rest of the system.



SAFETY CHECKS
pause if wrong

Watch-points run at every stage boundary. Most stop the round outright when something looks off — too little evidence, too many contradictions, a runaway loop — always at a clean boundary, never mid-task; a few only warn while they prove themselves.

EFFORT BUDGET
don't overrun

Each round gets a budget of time and parallel workers, not a dollar figure. A soft ceiling pauses for a decision before the limit, and a drift alarm catches cost climbing too fast and flags it early.

SAVE & RESUME
survive anything

The round's state is written at every step and locked so two machines can't clobber it. A crash or a hand-off to another machine mid-round picks up from the last completed step, not the start.

SEALED BOUNDARY
contain the web

Everything pulled from the open web is treated as untrusted and walled off: quarantined, never read as an instruction, with only cleared findings allowed into the record and no private data put in front of a web-facing agent.

Also running underneath: **a paywalled-paper fetcher** (legal sources first) · **the shared search tooling** all five scrapers draw on.

Figure (substrate). None of these belong to a single stage; they run across all of them. The result: a round can pause at any boundary, never loses more than the last step of work, resumes on another machine without restarting, and stays walled off from anything hostile it pulls in.

What these do

The stages do the research; this layer keeps it *safe, affordable, durable, and sealed*. The safety checks are the brakes, stopping a round that's going wrong before it pollutes the knowledge base. The budget is the throttle, holding back a round that would quietly burn a day's worth of compute. Save-and-resume is the seatbelt: an interruption costs a step, not the whole round. And the sealed boundary is the firewall: anything pulled from the open web is held behind it, treated as untrusted, never able to touch the rest of the system.

What's inside it

The watch-points — `most stop · some warn`

A dozen checks for the known ways an automated round goes wrong. Most can pause the round; the newer ones run as warnings first, and are promoted to full stops once they've shown they don't cry wolf.

The budget — `time × workers`

Measured in wall-clock and parallel workers rather than dollars, because the real limits here are rate and throughput. A soft ceiling pauses for a human; quality checks fire regardless of budget.

State & locks — `checkpoint`

Every boundary writes a checkpoint; a lock keeps two machines from writing at once; a paused round leaves a marker and waits for a go-ahead before continuing.

Containment — `untrusted by default`

Web content is wrapped and quarantined; every write to the record goes through a controller, not the agent that fetched it; and private data never enters a web-facing agent's prompt.

When they act

Continuously, at every stage boundary. A safety check or the budget can pause the round; a pause writes a marker and waits; a resume reads the last checkpoint and carries on. Because the checkpoint is written each step, the worst case is repeating one step, never losing the round.

SYNTHESIS · THE WHOLE SYSTEM

The system, in one read

It is a **general-purpose research engine**: a team of specialised agents, run as an adaptive loop, that any team can aim on demand at whatever needs researching — from a huge ambition down to a focused question. It carries that question from a vague request to a corroborated, buildable plan, casting a wide net, testing every claim from both sides, and looping back to research deeper whenever something won't hold. Because the work is deep, self-checking, and kept, it compounds the more it runs — and it only ever **proposes**; it hands off a plan, it never builds or deploys on its own.

Picture an assembly line for knowledge that loops back on itself: a question enters rough and moves down a line of specialists, each adding one thing well — and when what comes off the end isn't solid enough, it goes back up the line to be researched deeper, round after round, until it is.

THE PIPELINE, END TO END

01	**Intake**	Sits down with the request and pins down what is actually wanted, how deep, and by when.
02	**Plan**	Breaks the question into specific, budgeted search assignments a fixed set of agents can run in parallel.
03	**Background-brief**	Drops a short brief on every agent's desk, so each one starts already up to speed instead of cold.
04	**Gather**	Casts five wide nets at once — including one that deliberately fishes where the marketing-shaped sources will not.
05	**Verify**	Puts every claim on trial, argued both ways, and hands it back with a verdict and a stated confidence.
06	**Validate**	The border gate: only findings corroborated across independent sources are cleared into the permanent record.
07	**Gap-fill**	Walks the round for whatever is still thin and sends just those items back for one more bounded pass.
08	**Invention**	Turns the confirmed knowledge into concrete proposals — reusing a proven approach where one fits, inventing only where none does.
09	**Round-close**	Banks both what the round confirmed and what it learned about researching the topic, so the next round opens ahead.
·	**Throughout**	Trip-switches for failure modes, a budget on effort, save-and-resume across machines, and a sealed boundary against hostile content run under every stage.

WHAT TO TAKE AWAY

- **Nothing is trusted on a single source.** Every finding is corroborated and challenged before it is allowed to count.
- **It compounds the more it runs.** Each round banks what it learned, so the next one starts ahead of the last.
- **It delivers a plan; it never builds or deploys on its own.** Every output is a proposal for a person to weigh and build, not an action taken.
- **One operator gains the reach of a research team that does not forget.**
- **It retargets onto almost anything by preset, not by a rebuild** — from a huge ambition to a focused question, the same engine aimed somewhere new.

You've just seen how we think.

We run our own company on systems like this one. CASE builds them for businesses, from a website to a company's entire operating system. Most agencies stop where we start.

WEB
case.software

EMAIL
hello@case.software

INSTAGRAM
@case.software

X
@casesoftware

LINKEDIN
/company/case-software